MÁSTER UNIVERSITARIO EN
INVESTIGACIÓN MATEMÁTICA

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Resolución de ecuaciones de convección-difusión en 2D usando el método de las diferencias finitas compactas

Trabajo fin de máster

Presentado por
Ivan Peinado Asensi

Dirigido por
María Jezabel Pérez Quiles y Sergio Hoyas Calvo

Valencia, 09 de septiembre de 2019

Índice general

Resumen

Para el cálculo numérico de ecuaciones en derivadas parciales se han ido desarrollando múltiples métodos a los largo de los años. Las diferencias finitas compactas (CFD, Lele 1990) surgen como una actualización de las diferencias finitas y los esquemas interpolantes de Padé. Mediante el uso de moléculas en la derivada y en la función se consiguen fórmulas para la resolución de ecuaciones en derivadas parciales de orden muy alto con relativamente pocos puntos y en forma de matrices dispersas.

Por otro lado, en ingeniería, para el estudio de la aerodinámica alrededor de objetos sólidos se han usado desde hace más de 100 años los túneles de viento, con esto se han conseguido grandes avances en la aerodinámica y las disciplinas de la ingeniería aeronáutica pero a menudo limitados por el volumen del túnel y la velocidad del aire que podría entregarse, además del coste que implica la construcción de un túnel del tamaño adecuado a la dimensión del sólido y la maqueta que se quiere analizar.

Este proyecto está relacionado con un código en desarrollo en el IUMPA que pretende realizar simulaciones numéricas directas de objetos de túneles de viento virtuales mediante la técnica de Inmerse Boundary Conditions. Esto requiere técnicas computacionales extremadamente fiables y rápidas. Aunque el método CFD ha sido muy usado en problemas 1D, no se ha usado en 2D por motivos de sobre o sub determinación en los coeficientes de moléculas de dos dimensiones. En este TFM planteamos la posibilidad de hacer problemas 3D mediante el uso combinado de CFD en dos direcciones junto con un método de Fourier en la tercera, para así poder recrear el método de resolución comentado anteriormente.

Resum

Per al càlcul numèric décuacions en derivades parcials s'han desenvolupat múltiples mètodes al llarg dels anys, les diferències finites compactes (CFD, Lele 1990) sorgeixen com una actualització de les diferències finites i els esquemes interpolants de Padé. Mitjançant l'ús de molècules en la derivada i en la funció sáconsegueixen fórmules per a la resolució déquacions en derivades parcials dórdre molt alt amb relativament pocs punts i en forma de matrius disperses.

Dáltra banda, en ingenyeria, per a léstudi de láerodinàmica al voltant dóbjectes sòlids s'han usat desde fa més de 100 anys els túnes de vent, amb aixoà s'han aconseguit grans avanços en láerodinàmica i les disciplines de lénginyeria aeronàutica però sovint limitats pel volum del túnel i la velocitat de láire que podria entregar-se, a més del cost que implica la construcció dún túnel de la grandària adequada a la dimensió del sòlid i la maqueta que es vol analitzar.

Aquest projecte está relacionat amb un codi en desenvolupament en el IUMPA que pretén realitzr simulacions numèiques directes dóbjectes de túnels de vent virtuals mitjançant la tècnica de Inmerse Boundary Conditions. Això requereix tècniques computacionals extremadament fiables i ràpides. Encara que el mètodo CFD ha sigut molt usat en problemes 1D, no s'ha usat en 2D per motius de sobre o sub determinació en els coeficients de moloècules de dues dimensions. En aquest TFM plantegem la possibilitat de fer problemes 3D mitjançant l'ús combinat de CFD en dues direccions conjuntament amb un mètode de Fourier per a la tercera, per a així poder recrear el mètode de resolució comentat anteriorment.

Abstract

For the numerical calculation of partial diferential equations, several methods have been developed over the years. The compact finite differences (CFD, Lele 1990) arise as an update of the finite differences and interpolating Padé schemes. By using molecules in the derivative and in the function, formulas for solving equations in partial derivatives of very high order with relatively few points and in the form of dispersed matrices are achieved.

On the other hand, in engineering, for the study of aerodynamics around solid objects, wind tunnels have been used for more than 100 years, with this great advances have been achieved in aerodynamics and aeronautical engineering disciplines but to often limited by the volume of the tunnel and the speed of the air that could be delivered, in addition to the cost involved in the construction of a tunnel of adequate size to the size of the solid and the model to be analyzed.

This project is related to a code under development at IUMPA that aims to perform direct numerical simulations of virtual wind tunnel objects using the Inmerse Boundary Conditions technique. This requires extremely reliable and fast computational techniques. Although the CFD method has been widely used in 1D problems, it has not been used in 2D for reasons of over or under determination in the coefficients of two-dimensional molecules. In this TFM we propose the possibility of 3D problems through the combined use of CFD in two directions and together with a Fourier method in the third, in order to recreate the resolution method discussed above.

Índice de figuras

Capítulo 1

Introducción

La aerodinámica es una rama de la mecánica de fluido ampliamente estudiada. Los pioneros en este campo fueron los hermanos Wright, realizaron un vuelo controlado en 1903. Los hermanos empezaron a cuestionarse la veracidad de los datos aerodinámicos que estaban utilizando y crearon su propio túnel de viento y recogieron los datos más detallados en su momento para diseñar las alas de los aviones.

Desde entonces la tecnología para la recogida de datos aerodinámicos ha avanzado mucho. Desde túneles de viento de gran envergadura y diferentes sistemas para recoger los datos aerodinámicos, hasta la simulación en ordenadores de las ecuaciones que describen el movimiento de los fluidos. Uno de los grandes inconvenientes para obtener datos fiables en los túneles de viento eran los situados cerca de las paredes del sólido, ya que se integraban sensores en el sólido que se quería analizar y esto modificaba sensiblemente la superficie del sólido. Esto se ha solucionado con las simulaciones computacionales, donde se puede diseñar la superficie como se desee y obtener buenos resultados con el análisis adecuado.

Figura 1.1: Tunel de viento de los hermanos Wright

Lo que se pretende en este trabajo es realizar un código test para las ecuaciones de Navier-Stokes. Este código se va a testear con las ecuaciones de convección-difusión por su sencillez. Si se obtienen buenos resultados concluiremos que será valido para obtener resultados con las ecuaciones de modelización de los fluidos. A modo de objetivo personal, se pretende aprender los métodos numéricos y las técnicas computacionales para realizar dichos cálculos.

1.1. Métodos numéricos

El cálculo numérico es un área de estudio dentro del análisis matemático que permite encontrar una solución aproximada de problemas que describen fenómenos físicos aplicando métodos numéricos a ecuaciones en diferencias.

La estrategia general empleada para la resolución de un problema físico gobernado por una ecuación del tipo

$$\frac{\partial \mathbf{u}}{\partial t} + \mathcal{L}(\mathbf{u}) = 0, \tag{1.1}$$

consiste en discretizar el operador diferencial $\mathcal{L}$, que contiene las derivadas espaciales para convertirlo en una ecuación diferencial ordinaria en el tiempo. Esta estrategia recibe el nombre del método de las características. Existen distintas versiones de estos esquemas para conseguir expresar las derivadas espaciales como un operador algebraico.

1. **Diferencias finitas**. Las derivadas espaciales se aproximan por operadores algebraicos que se obtienen de series truncadas de Taylor. De esta forma, convertimos el operador $\mathcal{L}$ en una expresión algebraica, función de los valores de $\mathbf{u}$ en los puntos de una malla espacial. El error cometido en la discretización proviene de los términos de truncados en la serie de Taylor.

2. **Volúmenes finitos**. Planteamos la ecuación 1.1 en forma integral en vez de en forma diferencial. Definimos en el espacio un conjunto de volúmenes finitos, sobre los que imponemos el cumplimiento de la ecuación integral. En cada volumen finito se calcula el flujo a través de las fronteras, que será empleado en el cálculo de los volúmenes adyacentes como condición de contorno. Estos esquemas permiten trabajar con funciones que no son suficientemente regulares como para aplicar diferencias finitas.

3. **Elementos finitos**. Basada en la formulación débil de la ecuación 1.1, intenta expresar la solución como un vector sobre un espacio de funciones. Dependiendo de la forma de estas funciones se obtienen las distintas variables del método de los elementos finitos. Generalmente, estas funciones tienen carácter nodal; son nulas en todo el dominio salvo en las inmediaciones del nodo que representan.

4. **Diferencias finitas compactas**. Es en el método que nos vamos a basar en el trabajo, similar a las diferencias finitas, pero con relación de los valores de la derivada en varios nodos con el valor de la variable en los mismos nodos. De esta forma, para obtener la derivada se debe de resolver un sistema n-diagonal de ecuaciones algebraicas. En [9] se presentan distintos esquemas de diferencias finitas compactas que usaremos para aproximar las derivadas en nuestra ecuación.

5. **Métodos espectrales**.Se basan en aproximar la solución por su desarrollo en una base de un determinado espacio funcional. Esta base debe poseer ciertas características que faciliten el cálculo de las soluciones. La principal ventaja de estos métodos es la precisión espectral. Y algunos de estos métodos usan de series de Fourier o Chebyshev. De esta forma, aproximada la función, sus derivadas se obtienen de forma exacta. Su principal inconveniente es que los puntos de discretización vienen dados directamente por el método y que las geometrías a resolver deben ser extraordinariamente simples. En el presente proyecto utilizaremos Fourier.

Capítulo 2

Ecuación convección-difusión

Este trabajo va a consistir en resolver las ecuaciones de convección-difusión, son del tipo parabólica y se escriben tal que,

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} - \Delta \mathbf{u} + \vec{v}\,\nabla \mathbf{u} = f(x,y,t), \\ \dfrac{\partial \mathbf{v}}{\partial t} - \Delta \mathbf{v} + \vec{u}\,\nabla \mathbf{v} = g(x,y,t). \end{cases} \tag{2.1}$$

tomamos la primera ecuación con la cual empezaremos a trabajar antes de empezar a resolver el sistema, que escrita de forma extendida quedaría:

$$\frac{\partial \mathbf{u}}{\partial t} - \left(\frac{\partial^2 \mathbf{u}}{\partial x^2} + \frac{\partial^2 \mathbf{u}}{\partial y^2}\right) + \vec{v}\left(\frac{\partial \mathbf{u}}{\partial x} + \frac{\partial \mathbf{u}}{\partial y}\right) = f(x,y,t). \tag{2.2}$$

Para encontrar la solución a la ecuación se van a aplicar diferentes métodos numéricos que veremos en el siguiente capítulo, obteniendo un método constructivo que nos permitirá obtener una solución. Tradicionalmente para obtener el resultado de este tipo de ecuaciones el método de diferencias finitas compactas se ha aplicado en una dirección de la siguiente manera. Para hacer la discretización en el tiempo se utiliza el método de Runge-Kutta y la discretización espacial con diferencias finitas compactas para el eje x y con la transformada de Fourier para el eje y.

En este trabajo se va a intentar la resolución de la ecuación aplicando diferencias finitas compactas en las dos direcciones espaciales, además del método de Runge-Kutta para la discretización espacial. Una vez obtenido el resultado aplicaremos Fourier en la dirección z para poder obtener los datos del comportamiento del fluido sobre un solido. Los códigos realizados aparecen en el apartado Anexo I.

2.1. Resumen esquema numérico

El procedimiento a seguir para resolver el problema tiene su dificultad a la hora de elegir el método para conseguir la expresión 1.1 y definir apropiadamente la función

$\mathcal{L}(\mathbf{u})$. Hay numerosos métodos para resolver el problema planteado en este trabajo, en el presente proyecto aplicaremos el siguiente esquema numérico;

Para la discretización espacial:

- Para las derivadas en x e y se va a utilizar el método de diferencias finitas compactas [9]. Este método facilita aplicar las condiciones de contorno y presenta una mayor precisión que el método de diferencias finitas habitual.
- Para las derivadas en z, dirección periódica, se va a emplear el método espectral de la serie de Fourier, pues este facilita la derivación e integración numérica, además el coste computacional es menor que con otros métodos.

Y para la discretización temporal:

- Se va a aplicar un método de Runge-Kutta de 3 pasos que fue propuesto por Spalart *et al* en [11]. Este método es muy óptimo en el balance entre el coste computacional y el error acumulado para la obtención del resultado. Es un sistema semi-implícito, esto conlleva a resolver un sistema en cada paso.
  - Los términos lineales se incluirán en la parte implícita del sistema a resolver.
  - Los términos no-lineales serán explícitos, es decir, formarán parte del lado derecho de la ecuación.

En el siguiente capítulo se va a desarrollar teóricamente en que consiste los métodos que vamos a utilizar para crear nuestro código y encontrar la solución de la ecuación.

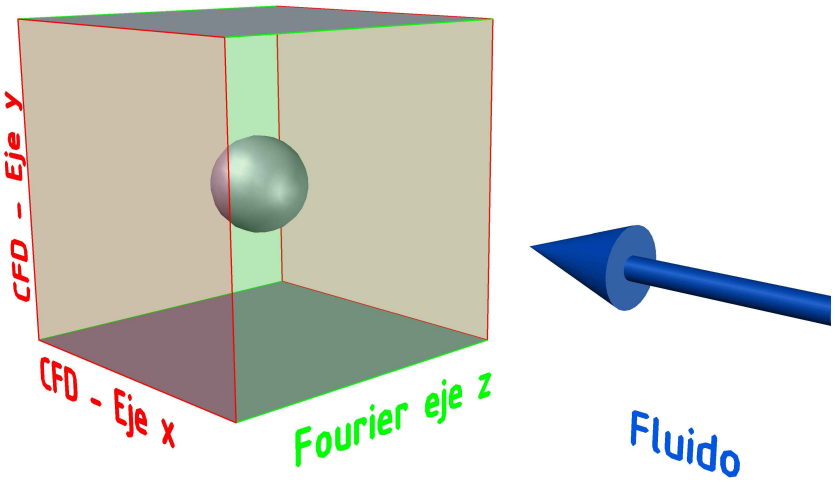

Figura 2.1: Detalle esquema numérico

Capítulo 3

Métodos numéricos

3.1. Diferencias finitas compactas

El método de diferencias finitas es ampliamente utilizado en problemas de contorno, basado en aproximar la derivada mediante una combinación lineal de los valores de la función en nodos contiguos, de modo que su adquisición es inmediata. No obstante, en el código implementaremos una versión más precisa del método, llamada diferencias finitas compactas (CFD), propuesto por S. K. Lele en 1990 en [9], en el artículo se observa que podemos utilizar este método hasta la k-ésima derivada, pero con nuestras ecuaciones solo vamos a utilizar hasta la segunda. En su aplicación se establece que, una combinación lineal de los valores en los nodos de la derivada de la función, es otra combinación lineal de los valores de la función en sí misma en los nodos. Esto implica que la resolución no es directa, sino que hay que resolver el sistema,

$$\mathbf{A}\mathbf{u}^{(k)} = \mathbf{B}\mathbf{u}. \tag{3.1}$$

El orden del método CFD es mucho mayor que con el habitual de diferencias finitas, y se suele aplicar solo para las diferencias en el eje y en el espacio físico. En la expresión genérica del método que acabamos de mostrar en la ecuación 3.1, las matrices $\mathbf{A}$ y $\mathbf{B}$ son las incógnitas del método, las cuales multiplican a les vectores $\mathbf{u}^{(k)}$ y $\mathbf{u}$ respectivamente. Estas se van a precisar para maximizar el orden del método, el cual se consigue resolviendo un problema compatible indeterminado, es decir, para un mismo error, existen infinitas soluciones.

Discretizamos el dominio N veces en dirección vertical y las condiciones de contorno

son impuestas en los nodos 1 y $N+1$.

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \\ u_{N+1} \end{pmatrix}; \qquad \mathbf{u}^k = \begin{pmatrix} u_1^{(k)} \\ u_2^{(k)} \\ \vdots \\ u_N^{(k)} \\ u_{N+1}^{(k)} \end{pmatrix}. \tag{3.2}$$

Los puntos están equiespaciados, como consecuencia, se van a poder obtenerse soluciones analíticas cerradas para $\mathbf{A}$ y $\mathbf{B}$.

3.1.1. Expresión genérica del método

En primer lugar se va a decidir el número de nodos contiguos que debemos tomar para calcular la derivada en un nodo en particular, es decir, el número de términos de la combinación lineal de la función original. Se asume que tomamos nodos consecutivos, $2N_1$ nodos vecinos para la función derivada y $2N_2$ nodos vecinos para la función en sí misma. La k-ésima derivada se expresa como:

$$\sum_{n=-N_1}^{N_1} \alpha_{i,i+n} \mathbf{u}_{i,i+n}^k = \sum_{n=-N_1}^{N_1} \beta_{i,i+n} \mathbf{u}_{i,i+n}, \tag{3.3}$$

donde los coeficientes $\alpha_{i,i+n}$ y $\beta_{i,i+n}$ forman las matrices $\mathbf{A}$ y $\mathbf{B}$ del siguiente modo:

$$\mathbf{A} = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \dots & \alpha_{1,N} \\ \alpha_{2,1} & \alpha_{2,2} & \dots & \alpha_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{N,1} & \alpha_{N,2} & \dots & \alpha_{N,N} \end{pmatrix}; \qquad \mathbf{B} = \begin{pmatrix} \beta_{1,1} & \beta_{1,2} & \dots & \beta_{1,N} \\ \beta_{2,1} & \beta_{2,2} & \dots & \beta_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{N,1} & \beta_{N,2} & \dots & \beta_{N,N} \end{pmatrix}. \tag{3.4}$$

Las matrices son diagonales, de tal forma que pueden resolverse mediante métodos preparados para que no sea necesario formarlas. $\mathbf{A}$ será N_1-diagonal y $\mathbf{B}$ será N_2-diagonal. Si se realiza un desarrollo en serie de Taylor, truncando en $H-k$ para la funcion derivada y en H para la función sin derivar, en torno a $\mathbf{u}_i$ y en torno a $\mathbf{u}_i^{(k)}$, la expresión 3.4 se puede formular de la siguiente forma,

$$\begin{aligned} \sum_{n=-N_1}^{N_1} \alpha_{i,i+n} \sum_{m=0}^{H-k} \frac{\Delta y_{(i+n,i)}^m}{m!} \frac{\delta^{k+m} u_i}{\delta y^{k+m} u_i} + \mathcal{O}(H-k+1) = \\ \sum_{n=-N_2}^{N_2} \beta_{i,i+n} \sum_{m=0}^{H} \frac{\Delta y_{(i+n,i)}^m}{m!} \frac{\delta^m u_i}{\delta y^m u_i} + \mathcal{O}(H+1). \end{aligned} \tag{3.5}$$

A partir de ahora vamos a asumir la ecuación anterior como exacta tras eliminas los términos $\mathcal{O}(H-k+1)$ y $\mathcal{O}(H+1)$ en ambos lados de la igualdad. Aunque de igual manera el orden del método seguirá siendo H que depende de N_1 y de N_2 tal que $H = 2(N_1+N_2)$. Volviendo a la expresión genérica, podemos agrupar los términos como combinaciones de la derivada p-ésima de u y $u_i^{(p)}$ a uno y otro lado de la igualdad. Estos términos deberán ser iguales en cada lado, para que se cumpla la relación 3.5 anterior independientemente del valor de $u_i^{(p)}$. De este modo se llega a un sistema de la forma

$$\begin{aligned}
0 &= \sum_{n=-N_2}^{N_2} \beta_{i,i+n}, \\
0 &= \sum_{n=-N_2}^{N_2} \beta_{i,i+n} \frac{\Delta y^1_{(i+n,i)}}{1!}, \\
&\vdots \\
0 &= \sum_{n=-N_2}^{N_2} \beta_{i,i+n} \frac{\Delta y^{k-1}_{(i+n,i)}}{(k-1)!}, \\
\sum_{n=-N_1}^{N_1} \alpha_{i,i+n} &= \sum_{n=-N_2}^{N_2} \beta_{i,i+n} \frac{\Delta y^{k}_{(i+n,i)}}{(k)!}, \\
\sum_{n=-N_1}^{N_1} \alpha_{i,i+n} \frac{\Delta y^1_{(i+n,i)}}{1!} &= \sum_{n=-N_2}^{N_2} \beta_{i,i+n} \frac{\Delta y^{k+1}_{(i+n,i)}}{(k+1)!}, \\
&\vdots \\
\sum_{n=-N_1}^{N_1} \alpha_{i,i+n} \frac{\Delta y^{H-k}_{(i+n,i)}}{(H-k)!} &= \sum_{n=-N_2}^{N_2} \beta_{i,i+n} \frac{\Delta y^{H}_{(i+n,i)}}{H!},
\end{aligned} \tag{3.6}$$

donde el conjunto de ecuaciones compone un sistema compatible indeterminado en los coeficientes α y β, para resolverlo es común imponer el valor unitario del coeficiente que acompaña al propio nodo en su derivada, $\alpha_{i,i+n}$. A pesar de que este término parece tener influencia en las ecuaciones 4 y 6 del sistema 3.6, la realidad es que únicamente puede considerarse en la cuarta ecuación. Esto se debe a que en el resto de términos aparece multiplicada por el término $\Delta y_{(i,j)}$, anulándose porque dicha distancia, es decir, la separación entre un nodo y él mismo, es nula. Con dicha suposición la cuarta ecuación nos queda,

$$1 = \sum_{n=-N_2}^{N_2} \beta_{i,i+n} \frac{\Delta y^k_{i+n,i}}{k!} - \sum_{n=-N_1}^{N_1} \alpha_{i,i+n}, \qquad \text{con } n \neq 0. \tag{3.7}$$

Con esto queda un sistema compatible determinado, de tamaño $H+1$, de tal forma que se resuleve $2J_1$ términos de α, y $(2J_2+1)$ términos de β. El sistema no es homogéneo debido a que se impone la expresión anterior 3.7. En los siguientes apartados veremos

como obtener la matriz para los casos de estudio de la segunda derivada, tanto en el centro del dominio como cerca de los contornos.

3.1.2. Matriz de coeficientes para la segunda derivada

Para la segunda derivada, $k = 2$ se considera el valor de 2 nodos vecinos tanto para la segunda derivada como para la función sin derivar, es decir, $N_1 = N_2 = 2$, de esta forma $H = 8$, que junto con el nodo k-ésimo se nos queda un sistema de 9 ecuaciones con 9 incógnitas ($\alpha_{i,i-2}$, $\alpha_{i,i-1}$, $\alpha_{i,i+1}$, $\alpha_{i,i+2}$, $\beta_{i,i-2}$, $\beta_{i,i-1}$, $\beta_{i,i}$, $\beta_{i,i+1}$, $\beta_{i,i+2}$) para cada nodo i situado en una determinada posición vertical.

$$
\begin{aligned}
\sum_{n=-2}^{2} \beta_{i,i+n} &= 0, \\
\sum_{n=-2}^{2} \beta_{i,i+n}\, \Delta y_{i+n,i} &= 0, \\
\sum_{n=-2}^{2} \beta_{i,i+n} \frac{\Delta y_{i+n,i}^2}{2} - \sum_{n=-2}^{2} \alpha_{i,i+n} &= 1, \\
\sum_{n=-2}^{2} \beta_{i,i+n} \frac{\Delta y_{i+n,i}^3}{3!} - \sum_{n=-2}^{2} \alpha_{i,i+n}\, \Delta y_{i+n,i} &= 0, \\
\sum_{n=-2}^{2} \beta_{i,i+n} \frac{\Delta y_{i+n,i}^4}{4!} - \sum_{n=-2}^{2} \alpha_{i,i+n} \frac{\Delta y_{i+n,i}^2}{2} &= 0, \\
&\vdots \\
\sum_{n=-2}^{2} \beta_{i,i+n} \frac{\Delta y_{i+n,i}^8}{8!} - \sum_{n=-2}^{2} \alpha_{i,i+n} \frac{\Delta y_{i+n,i}^6}{6!} &= 0,
\end{aligned}
\tag{3.8}
$$

con $n \neq 0$. Este sistema es compatible determinado y resoluble, no obstante, suele presentar problemas de mal condicionamiento, lo cual se soluciona multiplicando cada fila por el factorial máximo de esta, y dividiendo por el Δy^p mínimo de la misma. Cuando el nodo i-ésimo se encuentra cerca de un contorno, el sistema no estaría definido, pues haría uso de distancias que saldrían del dominio. Para solucionar esto, se debe disminuir el orden del sistema, pues de orden $H = 8$ pasar a $H = 4$, variando los nodos que se van a tomar. Para el nodo $i = 2$ se toma $N_1 = 1$ y $N_2 = 1$, y el sistema

quedaría con 5 ecuaciones y 5 incógnitas ($\alpha_{2,1}$, $\alpha_{2,3}$, $\beta_{2,1}$, $\beta_{2,2}$, $\beta_{2,3}$) tal que

$$\begin{aligned}
\beta_{2,1} + \beta_{2,2} + \beta_{2,3} &= 0, \\
\beta_{2,1}\Delta y_{1,2} + \beta_{2,3}\Delta y_{3,2} &= 0, \\
\frac{1}{2}\left(\beta_{2,1}\Delta y_{1,2}^2 + \beta_{2,3}\Delta y_{3,2}^2\right) - (\alpha_{2,1} + \alpha_{2,3}) &= 1, \\
\frac{1}{3!}\left(\beta_{2,1}\Delta y_{1,2}^3 + \beta_{2,3}\Delta y_{3,2}^3\right) - (\alpha_{2,1}\Delta y_{1,2} + \alpha_{2,3}\Delta y_{3,2}) &= 0, \\
\frac{1}{4!}\left(\beta_{2,1}\Delta y_{1,2}^4 + \beta_{2,3}\Delta y_{3,2}^4\right) - \frac{1}{2}(\alpha_{2,1}\Delta y_{1,2}^2 + \alpha_{2,3}\Delta y_{3,2}^2) &= 0.
\end{aligned} \tag{3.9}$$

Para el nodo $i = N-1$, el desarrollo es totalmente análogo al anterior. Por contra, para el nodo $i = 1$ o $i = N$ el sistema será diferente. Se tomarán dos nodos contiguos para calcular dichos puntos, como en el caso previo, pero para $i = 1$ se tomarán $i = 2, 3$ y para $i = N$, $i = N-2$, $N-1$ y ahora las incógnitas serán ($\alpha_{1,2}$, $\alpha_{1,3}$, $\beta_{1,1}$, $\beta_{1,2}$, $\beta_{1,3}$). El sistema resultante es el siguiente

$$\begin{aligned}
\beta_{1,1} + \beta_{1,2} + \beta_{1,3} &= 0, \\
\beta_{1,2}\Delta y_{2,1} + \beta_{1,3}\Delta y_{3,1} &= 0, \\
\frac{1}{2}\left(\beta_{1,2}\Delta y_{2,1}^2 + \beta_{1,3}\Delta y_{3,1}^2\right) - (\alpha_{1,2} + \alpha_{1,3}) &= 1, \\
\frac{1}{3!}\left(\beta_{1,2}\Delta y_{2,1}^3 + \beta_{1,3}\Delta y_{3,1}^3\right) - (\alpha_{1,2}\Delta y_{2,1} + \alpha_{1,3}\Delta y_{3,1}) &= 0, \\
\frac{1}{4!}\left(\beta_{1,2}\Delta y_{2,1}^4 + \beta_{1,3}\Delta y_{3,1}^4\right) - \frac{1}{2}(\alpha_{1,2}\Delta y_{2,1}^2 + \alpha_{1,3}\Delta y_{3,1}^2) &= 0.
\end{aligned} \tag{3.10}$$

En resumen, para iniciar el método de las diferencias finitas compactas en el código, se deberá realizar antes de iniciar el problema lo siguiente:

- Resolver el sistema de ecuaciones 3.8 que nos permita obtener las incógnitas $\alpha_{i,i-2}$, $\alpha_{i,i-1}$, $\alpha_{i,i+1}$, $\alpha_{i,i+2}$, $\beta_{i,i-2}$, $\beta_{i,i-1}$, $\beta_{i,i}$, $\beta_{i,i+1}$, $\beta_{i,i+2}$ además de $\alpha_{i,i} = 1$. Válido para la discretización $3 \leq i \leq N-2$.

- Resolver el sistema de ecuaciones 3.9 que nos permita obtener las incógnitas $\alpha_{2,1}$, $\alpha_{2,3}$, $\beta_{2,1}$, $\beta_{2,2}$, $\beta_{2,3}$ para $i = 2$ y de forma análoga para $i = N-1$ obtener $\alpha_{N-1,N-2}$, $\alpha_{N-1,N}$, $\beta_{N-1,N-2}$, $\beta_{N-1,N-1}$, $\beta_{N-1,N}$. Asimismo $\alpha_{2,2} = \alpha_{N-1,N-1} = 1$.

- Resolver el sistema de ecuaciones 3.10 que nos permita obtener las incógnitas $\alpha_{1,2}$, $\alpha_{1,3}$, $\beta_{1,1}$, $\beta_{1,2}$, $\beta_{1,3}$ para $i = 1$ y de forma análoga para $i = N$ obtener $\alpha_{N,N-2}$, $\alpha_{N,N-1}$, $\beta_{N,N-2}$, $\beta_{N,N-1}$, $\beta_{N,N}$. Asimismo $\alpha_{1,1} = \alpha_{N,N} = 1$.

Con todo esto, ya tenemos los elementos necesarios para obtener las matrices pentadiagonales del sistema como muestra la ecuación 3.4 y así obtener la herramienta final para resolver el sistema 3.1 en la segunda derivada:

$$\mathbf{A}\frac{\partial^2 \mathbf{u}}{\partial y^2} = \mathbf{B}\mathbf{u} \tag{3.11}$$

3.1.3. Matriz de coeficientes para la primera derivada

El procedimiento para obtener las matrices **C** y **D** que satisfacen,

$$\mathbf{C}\frac{\partial \mathbf{u}}{\partial y} = \mathbf{D}\mathbf{u}, \tag{3.12}$$

es análogo al de la sección anterior. Se parte de las ecuaciones vitas en 3.6 con $k = 1$ y $N_1 = N_2 = 3$, siendo el orden del método $H = 12$. Para los puntos cercanos de la frontera se han empleado métodos de orden 10, 8 y 6 conforme nos acercábamos al contorno. Para $4 \leq i \leq N-3$ dónde $\alpha_{i,i} = 1$ e incógnitas $\alpha_{i,i-3}$, $\alpha_{i,i-2}$, $\alpha_{i,i-1}$, $\alpha_{i,i+1}$, $\alpha_{i,i+2}$, $\alpha_{i,i+3}$, $\beta_{i,i-3}$, $\beta_{i,i-2}$, $\beta_{i,i}$ $\beta_{i,i+1}$, $\beta_{i,i+2}$, $\beta_{i,i+3}$, se formula el siguiente sistema,

$$\begin{aligned}
\sum_{n=-3}^{3} \beta_{i,i+n} &= 0, \\
\sum_{n=-3}^{3} \beta_{i,i+n}\Delta y_{i+n,i} - \sum_{n=-3}^{3} \alpha_{i,i+n} &= 1, \\
\sum_{n=-3}^{3} \beta_{i,i+n}\frac{\Delta y_{i+n,i}^2}{2} - \sum_{n=-3}^{3} \alpha_{i,i+n}\,\Delta y_{i+n,i} &= 0, \\
\sum_{n=-3}^{3} \beta_{i,i+n}\frac{\Delta y_{i+n,i}^3}{3!} - \sum_{n=-3}^{3} \alpha_{i,i+n}\frac{\Delta y_{i+n,i}^2}{2} &= 0, \\
&\vdots \\
\sum_{n=-3}^{3} \beta_{i,i+n}\frac{\Delta y_{i+n,i}^{10}}{10!} - \sum_{n=-3}^{3} \alpha_{i,i+n}\frac{\Delta y_{i+n,i}^{9}}{9!} &= 0, \\
\sum_{n=-3}^{3} \beta_{i,i+n}\frac{\Delta y_{i+n,i}^{11}}{11!} - \sum_{n=-3}^{3} \alpha_{i,i+n}\frac{\Delta y_{i+n,i}^{10}}{10!} &= 0.
\end{aligned} \tag{3.13}$$

Y los sistemas para los puntos cercanos a las fronteras se obtendrían a partir de este sistema de igual manera que hemos visto en la sección anterior, realizando lo mismo que en las ecuaciones 3.9 y 3.10. El orden del método para la primera derivada es superior a la segunda, debido a que las matrices de la primera derivada se emplean para derivas y las de la segunda derivada se emplean para integrar, esto es debido a el coste computacional y los errores numéricos al derivar son mayores que cuando integramos, por eso el primer esquema 3.13 es más preciso.

3.1.4. Derivación y resolución de ecuaciones diferenciales

Para calcular las derivadas de las ecuaciones 3.11 y 3.12 es posible hacerlo de forma directa tal que

$$\frac{\partial^2 \mathbf{u}}{\partial y^2} = \mathbf{A}^{-1}\mathbf{B}\mathbf{u}; \qquad \frac{\partial \mathbf{u}}{\partial y} = \mathbf{C}^{-1}\mathbf{D}\mathbf{u}. \tag{3.14}$$

No obstante, el auténtico potencial de las diferencias finitas compactas reside en resolver ecuaciones diferenciales. Para ellos será necesario aplicar condiciones de contorno, o de lo contrario, el sistema será irresoluble. El código debe resolver la siguiente ecuación diferencial,

$$\frac{\partial^2 \mathbf{u}}{\partial y^2} - \lambda \mathbf{u} = f, \tag{3.15}$$

que la ecuación la vamos a escribir de manera simplificada por comodidad

$$\mathbf{u}''(y) - \lambda \mathbf{u}(y) = f(y). \tag{3.16}$$

Esta ecuación diferencial es de tipo elíptica con un coeficiente no lineal, comúnmente conocida como ecuación de Poisson. Aplicando los métodos de Fourier y Runge-Kutta, que veremos en las siguientes secciones, a la ecuación 3.15 ya no es en derivadas parciales, si no que es ordinaria. Ahora para resolverlo por el método de las diferencias finitas compactas se multiplica la ecuación 3.16 por la matriz $\mathbf{A}$ tal que

$$\mathbf{Au"} - \lambda \mathbf{Au} = \mathbf{A}f, \tag{3.17}$$

aplicando 3.11,

$$\mathbf{Bu} - \lambda \mathbf{Au} = \mathbf{A}f \quad \rightarrow \quad (\mathbf{B} - \lambda \mathbf{A})\mathbf{u} = \mathbf{A}f. \tag{3.18}$$

Ahora se podría calcular la inversa de $(\mathbf{B} - \lambda \mathbf{A})$ para obtener el valor de $\mathbf{u}$ pero no obtendríamos una solución correcta debido a que no hemos aplicado las condiciones de contorno ya que debemos recordar que la ecuación es solamente válida en el abierto, es decir, no podemos calcular la derivada en el contorno. Aplicamos la condiciones de contorno para que sea invertible y como es de segundo orden, aplicamos dos condiciones.

Condición de contorno Dirichlet

Se trata de resolver el siguiente sistema.

$$\begin{cases} \mathbf{u}'' - \lambda \mathbf{u} &= f, \\ \mathbf{u}(y_1) &= \alpha, \\ \mathbf{u}(y_N) &= \beta. \end{cases} \tag{3.19}$$

La forma de resolver el sistema es disociando el vector $\mathbf{u}$ en dos partes, una la incógnita que queremos resolver $\mathbf{u}_y$ y la otra la condición de contorno conocida $\mathbf{u}_{cc}$ tal

que

$$\mathbf{u}_y = \begin{pmatrix} 0 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \\ 0 \end{pmatrix}; \qquad \mathbf{u}_{cc} = \begin{pmatrix} \alpha \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \beta \end{pmatrix}; \qquad \mathbf{u} = \mathbf{u}_y + \mathbf{u}_{cc}. \tag{3.20}$$

Sustituimos $\mathbf{u} = \mathbf{u}_y + \mathbf{u}_{cc}$ en 3.18 y obtenemos

$$(\mathbf{B} - \lambda\mathbf{A})(\mathbf{u}_y + \mathbf{u}_{cc}) = \mathbf{A}f, \quad \rightarrow \quad (\mathbf{B} - \lambda\mathbf{A})\mathbf{u}_y = \mathbf{A}f - (\mathbf{B} - \lambda\mathbf{A})\mathbf{u}_{cc} \tag{3.21}$$

Ahora ya podemos resolver el sistema y obtener la solución del espacio previamente discretizado

$$\mathbf{u}_y = (\mathbf{B} - \lambda\mathbf{A})^{-1} \left[\mathbf{A}f - (\mathbf{B} - \lambda\mathbf{A})\mathbf{u}_{cc}\right] \tag{3.22}$$

3.2. Discretización espacial

De las posibles discretizaciones espaciales que se pueden aplicar al problema, como ya hemos nombrado, lo más común y eficiente que se está utilizando es la combinación del método CFD con métodos espectrales para cada dirección espacial. El principal motivo de esta elección es porque tiene una gran precisión, obteniendo aproximaciones excelentes con un número moderado de coeficientes.

3.2.1. Introducción a los métodos espectrales

Los métodos espectrales se caracterizan por el desarrollo en serie de la función $\mathbf{u}(x)$ mediante una base infinita de funciones ortogonales Φ_k tal que

$$\mathbf{u}(x) = \sum \hat{\mathbf{u}}_k \Phi_k. \tag{3.23}$$

Si seleccionamos adecuadamente las funciones de base Φ_k y si la función $\mathbf{u}(x)$ es infinitamente suave, el coeficiente k-ésimo del desarrollo decae más rápido que cualquier potencia inversa de k. En la práctica se comprueba que este comportamiento no se produce hasta que tenemos suficientes coeficientes para representar adecuadamente la estructura de la función $\mathbf{u}(x)$. Esta propiedad recibe el nombre de precisión espectral. Esta rápida caída a cero de los coeficientes del desarrollo en serie permite obtener aproximaciones excelentes con un número de coeficientes. Fourier solo es aplicable a problemas con condiciones de contorno periódicas.

3.2.2. Desarrollo en serie de Fourier

Para el desarrollo en serie de Fourier se seleccionan como funciones de base las exponenciales complejas tal que

$$\Phi_k(x) = e^{ikx} \tag{3.24}$$

Estas funciones, definidas en el intervalo $(0, 2\pi)$, son un conjunto de funciones ortogonales con el producto escalar,

$$\langle \Phi_k(x), \Phi_l(x) \rangle = \int_0^{2\pi} \Phi_k(x) \overline{\Phi_l(x)}\, dx, \tag{3.25}$$

$$\langle \Phi_k(x), \Phi_l(x) \rangle = 2\pi \delta_{(k,l)}, \tag{3.26}$$

donde $\delta_{(k,l)}$ es la delta de Kronecker y $\overline{\Phi}$ denota el conjunto de Φ. De esta forma se definen los coeficientes de Fourier de la función $\mathbf{u}$.

$$\hat{\mathbf{u}}_k = \int_0^{2\pi} \mathbf{u}(x) e^{-ikx} dx, \qquad k = -\infty, \ldots, -1, 0, 1, \ldots, \infty. \tag{3.27}$$

Estos coeficientes serán, en general, números complejos caracterizados por su módulo y su fase. Si representamos el módulo de los coeficientes en función de k obtendremos el espectro de amplitud. Representando la fase de coeficientes en función de k obtendremos el espectro de fase.

El desarrollo en serie de Fourier S_u de la función $\mathbf{u}(x)$ se convierte así en una serie infinita de términos tal que

$$S_u = \sum_{k=-\infty}^{\infty} \hat{\mathbf{u}}_k e^{ikx}. \tag{3.28}$$

El tratamiento numérico de esta serie obliga a truncarla para un número total de coeficientes N, dando lugar a la serie truncada de Fourier,

$$P_N = \sum_{k=-N/2}^{N/2-1} \hat{\mathbf{u}}_k e^{ikx}. \tag{3.29}$$

Cuestiones como condiciones de convergencia de esta serie truncada, la relación entre serie y la función o la velocidad de convergencia de la serie no serán tratadas en este proyecto. Basta decir que si la función es continua y periódica, el desarrollo en serie de Fourier es uniformemente convergente, aunque la convergencia puntual no está asegurada. Si la función no fuera periódica, se tomaría un periodo determinado y se extendería la función periódicamente fuera de dicho periodo.

3.2.3. Desarrollo discreto de Fourier

Al intentar dar un tratamiento numérico a las serie de Fourier nos encontramos con algunas dificultades. En primer lugar, los coeficientes de Fourier de una función dada no se conocen de forma exacta y deben de aproximarse de alguna forma. Por otro lado, se necesita una forma eficiente de recuperar en el espacio físico la información calculada en el espacio de Fourier. Para resolver estas dificultades se emplea la serie discreta de Fourier.

Consideremos un conjunto de N puntos definidos en el intervalo $(0,\, 2\pi)$, a los que llamaremos nodos,

$$x_j = \frac{2\pi j}{N}, \qquad j = 0, \ldots, N-1. \tag{3.30}$$

Se llaman coeficientes de la serie discreta de Fourier a los números complejos obtenidos de

$$\tilde{\mathbf{u}}_k = \frac{1}{N} \sum_{j=0}^{N-1} \mathbf{u}(x) e^{-ikx_j}. \tag{3.31}$$

La fórmula de inversión para estos coeficientes es,

$$\mathbf{u}(x_j) = \sum_{k=-N/2}^{N/2-1} \tilde{\mathbf{u}}_k e^{ikx_j}. \tag{3.32}$$

La obtención de cada coeficiente $\tilde{\mathbf{u}}_k$ requiere $O(N)$ operaciones, por lo que N coeficientes requieres $O(N^2)$ operaciones, la transformada rápida de Fourier (FFT) en su versión más sencilla es un algoritmo que reduce el número de operaciones necesaria a $O(N \log_2 N)$, pero impone la condición de que el número de coeficientes de qué condiciones se le impone a N y de si los coeficientes de la serie discreta de Fourier son reales o complejos. En nuestro caso utilizaremos la versión de *fast fourier transform in the west* (FFTW), que es la implementada en MatLab.

3.3. Discretización temporal

Después de llevar a cabo la discretización espacial del problema, hemos obtenido una ecuación diferencial de primer orden a partir la ecuación 1.1

$$\frac{\partial \mathbf{u}}{\partial t} + \mathcal{L}_h(\mathbf{u}) = 0, \tag{3.33}$$

donde $\mathcal{L}_h$ representa el operador diferencial que contiene las derivadas espaciales discretizadas.

Hay una amplia familia de los métodos numéricos que permiten la integración de esta ecuación, la clasificación de estos métodos se puede realizar atendiendo los siguientes criterios:

- Según el método de generación del esquema numérico, cuadratura numérica o diferenciación finita
- Según el número de pasos que involucra el esquema, métodos unipaso o métodos multipaso.
- Segun el carácter del sistema de ecuaciones resultante, métodos explícitos o métodos implícitos.

Una explicación detalla de los distintos métodos de integración y su aplicación a las ecuaciones se puede encontrar en [8].

3.3.1. Método de Runge-Kutta

La expresión general de los esquemas Runge-Kutta es,

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \sum_{i=1}^{s} b_i K_i. \tag{3.34}$$

donde los coeficientes k_i son términos de aproximación intermedios, evaluados en la función de manera local.

$$k_i = \mathcal{L}_h(u^n + \Delta t \sum_{i=1}^{s} a_{ij} K_j,\, t_n + c_i \Delta t), \quad i = 1, \ldots, s, \tag{3.35}$$

con a_{ij}, b_i y c_i coeficientes del esquema numérico dependiente de la regla de cuadratura utilizada. Si el coeficiente a_{ij} es nulo y $j \geq i$ se dice que es explícito, pues cada k_i depende de los k_i ya calculados, en cambio, cuando esto no ocurre el método es implícito, pues sería necesario resolver un sistema en cada paso. Los métodos de Runge-Kutta son una amplia familia de métodos empleado para resolver problemas de valor inicial, el esquema que vamos a utilizar en esta simulación es un esquema de tres subpasos, definido para funciones sin dependencia explícita con el timepo y que requiere dos niveles de almacenamiento. El esquema es de orden 3 para funciones lineales y 2 para funciones no lineales. La formulación que vamos a utilizar fue propuesta por P. R. Spalart *et all* [11]. Para avanzar desde $\mathbf{u}^n$ hasta $\mathbf{u}^{n+1}$ en el tiempo $t + \Delta t$ el método tiene 3 subpasos tal que,

$$\mathbf{u}_1^n = \mathbf{u}^n + \Delta t \left[L(\alpha_1 \mathbf{u}^n + \beta_1 \mathbf{u}_1^n) + \gamma_1 N^n \right], \tag{3.36}$$

$$\mathbf{u}_2^n = \mathbf{u}_1^n + \Delta t \left[L(\alpha_2 \mathbf{u}_1^n + \beta_2 \mathbf{u}_2^n) + \gamma_2 N_1^n + \zeta_1 N^n \right], \tag{3.37}$$

$$\mathbf{u}^{n+1} = \mathbf{u}_2^n + \Delta t \left[L(\alpha_3 \mathbf{u}_2^n + \beta_3 \mathbf{u}_3^n) + \gamma_3 N_2^n + \zeta_2 N_1^n \right], \tag{3.38}$$

donde L es el término lineal de la ecuación y N el no lineal tal que $N^n \equiv N(\mathbf{u}^n)$, $N_1^n \equiv N(\mathbf{u}_1^n)$, $N_2^n \equiv N(\mathbf{u}_2^n)$. Por tanto a partir de la solución conocida en el paso temporal n podemos obtener la solución en el siguente paso $\mathbf{u}^{n+1}$ tras aplicar los tres subpasos. El valor de los coeficientes para obtener una buena solución puede verse como se han obtenido en [11] y son los siguientes,

$$\begin{array}{llll} \gamma_1 = \dfrac{8}{15}, & \gamma_2 = \dfrac{5}{12}, & \gamma_3 = \dfrac{3}{4}, & \zeta_1 = -\dfrac{17}{60}, \quad \zeta_2 = -\dfrac{5}{12}, \\ \alpha_1 = \dfrac{29}{96}, & \alpha_2 = -\dfrac{3}{40}, & \alpha_3 = \dfrac{1}{6}, & \\ \beta_1 = \dfrac{37}{160}, & \beta_2 = \dfrac{5}{24}, & \beta_3 = \dfrac{1}{6}. & \end{array} \tag{3.39}$$

3.4. Condición de Courant-Friedrichs-Lewy (CFL)

El análisis de un esquema numérico incluye un estudio de la precisión del esquema, así como un análisis de estabilidad de la solución numérica. La precisión del esquema está asociada a la exactitud con que representamos la solución de las ecuaciones y se relaciona directamente con el orden del esquema empleado y con el número de puntos de la discretización espacial del dominio. La estabilidad de la solución numérica está relacionada con la amplificación o atenuación de errores al avanzar en el tiempo.

La estabilidad de la solución numérica se estudia mediante el método de Von Neuman, en el que una ecuación del tipo,

$$\frac{\partial \mathbf{u}}{\partial t} = F(\mathbf{u}), \tag{3.40}$$

se linealiza para convertirse en el problema de autovalores,

$$\frac{\partial \mathbf{u}}{\partial t} = \lambda(\mathbf{u}). \tag{3.41}$$

Estudiando si este autovalor λ cae dentro de la región de convergencia del esquema de integración temporal empleado, se puede determinar si la solución es estable o no. Puesto que el autovalor λ dependerá de la discretización espacial empleada, en la estabilidad de la solución numérica interactuarán la discretización espacial y temporal.

La condición de Courant-Friedrichs-Lewy, comúnmente llamada CFL, establece la relación entre la discretización espacial y la temporal para asegurar la estabilidad del esquema numérico, basándose en el concepto anterior. Esta condición se expresa matemáticamente como,

$$\text{CFL} \leq 1, \qquad \text{CFL} = \pi \frac{u\Delta t}{\Delta x}, \tag{3.42}$$

donde u es la velocidad, Δx la discretización de la malla en dirección de u y Δt es el paso en el tiempo. La interpretación física de esta condición es que no podemos permitir que ninguna partícula fluida avance en un paso de tiempo una distancia mayor a la resolución de la malla en ese punto. Hay que tener en cuenta, que en general, el CFL no es constante en todo el dominio de integración, provocando que la solución pueda ser inestable en unas zonas mientras que en otras no.

Partimos de la base de que un un buen CFL para obtener el resultado que buscamos es $CFL = 0{,}7$ y el Δx viene definido por la maya, por tanto, a partir de esto obtenemos el Δt máximo que podemos utilizar en el problema.

Capítulo 4

Aplicación del esquema numérico

En este capítulo veremos como se ha aplicado el esquema numérico a la ecuación de convección-difusión 2.1 que la escribiremos de manera reducida tal que,

$$\mathbf{u}_t - (\mathbf{u}_{xx} + \mathbf{u}_{yy} + \mathbf{u}_{zz}) + \vec{u}(\mathbf{u}_x + \mathbf{u}_y + \mathbf{u}_z) = f(x, y, z, t). \tag{4.1}$$

El esquema numérico para resolver la ecuación consta de los métodos de diferencias finitas compactas 2D en los ejes x e y, espectral por Fourier en el eje z y Runge-Kutta para el tiempo.

4.1. Diferencias finitas compactas en 2D

En la siguiente sección se presenta como hemos aplicando el método de DFC en 2D. Para resolver la ecuación vamos a partir del planteamiento de un problema típico en matemática algebraica tal que,

$$\mathbf{A}\mathbf{u} = f, \tag{4.2}$$

donde

$$\mathbf{u} = \begin{pmatrix} 1 \\ \vdots \\ N \end{pmatrix}, \tag{4.3}$$

pero en nuestro caso la $\mathbf{u}$ es una matriz de tamaño $N \times M$

$$\mathbf{u} = \begin{pmatrix} 1 & \dots & M \\ \vdots & & \vdots \\ N & \dots & NM \end{pmatrix} \tag{4.4}$$

Por tanto vamos a plantear el problema a partir de que,

$$\mathbf{A}_x \mathbf{u} \mathbf{A}_y = f. \tag{4.5}$$

Con este planteamiento ya podemos abordar como resolver la ecuación de convección-difusión con las matrices obtenidas por el método de DFC.

4.1.1. En derivada segunda

Primero vamos a ver como se aplica a la segunda derivada, con las matrices obtenidas a partir de le código que se nos ha proporcionado procedemos de la siguiente manera. Partimos del problema de Poisson para calcular la segunda derivada,

$$-\nabla^2 \mathbf{u} = f(x, y). \tag{4.6}$$

Procedemos multiplicamos todo por la matriz $\mathbf{A}_{xx}$.

$$-(\mathbf{A}_{xx}\partial_{xx} + \mathbf{A}_{xx}\partial_{yy})(\mathbf{u}) = \mathbf{A}_{xx} f(x, y), \tag{4.7}$$

y a continuación multiplicamos por la matriz A_{yy},

$$-(\mathbf{A}_{xx}\partial_{xx}\mathbf{A}'_{yy} + \mathbf{A}_{xx}\partial_{yy}\mathbf{A}'_{yy})(\mathbf{u}) = \mathbf{A}_{xx} f(x, y)\mathbf{A}'_{yy} \tag{4.8}$$

Multiplicamos por la transpuesta de $\mathbf{A}_{yy}$ para que coincidan los índices obtenidos en cada dirección. Como hemos visto en el capítulo anterior, las matrices que obtenemos del método de DFC cumplen la igualdad $\mathbf{A}_{xx}\mathbf{u} = \mathbf{B}_{xx}\mathbf{u}_{xx}$ donde $\mathbf{A}_{xx}$ y $\mathbf{B}_{xx}$ son las matrices que obtenemos del código para calcular la derivada segunda. Por tanto para ambas direcciones,

$$\begin{array}{rcl} \mathbf{A}_{xx}\partial_{xx} & = & \mathbf{B}_{xx}\mathbf{u}, \\ \mathbf{A}_{yy}\partial_{yy} & = & \mathbf{B}_{yy}\mathbf{u}, \end{array} \tag{4.9}$$

de la expresión 4.8 llegamos a,

$$-\mathbf{B}_{xx}\mathbf{u}(x, y)\mathbf{A}'_{yy} - \mathbf{A}_{xx}\mathbf{u}(x, y)\mathbf{B}'_{yy} = \mathbf{A}_{xx} f(x, y)\mathbf{A}'_{yy}. \tag{4.10}$$

Para mayor comodidad denotaremos la operación de las matrices $\mathbf{AuB}$ con el operador $\mathbf{A} \otimes \mathbf{B}$ y quedaría,

$$-(\mathbf{B}_{xx} \otimes \mathbf{A}'_{yy} + \mathbf{A}_{xx} \otimes \mathbf{B}'_{yy})(\mathbf{u}) = \mathbf{A}_{xx} f(x, y)\mathbf{A}'_{yy}. \tag{4.11}$$

Ahora podemos obtener el resultado de $\mathbf{u}$ haciendo la inversa de las matrices de la parte izquierda y multiplicarla por la parte derecha que es conocida,

$$\mathbf{u} = (-\mathbf{B}_{xx} \otimes \mathbf{A}'_{yy} - \mathbf{A}_{xx} \otimes \mathbf{B}'_{yy}) \backslash (\mathbf{A}_{xx} f(x, y)\mathbf{A}'_{yy}). \tag{4.12}$$

Pero una vez llegado a este punto tenemos el problema de multiplicar tres matrices $\mathbf{A} \otimes \mathbf{B}$ con $\mathbf{u}$ como incógnita. Por tanto hay que crear una matriz $\mathbf{AA}$ que no conocemos el orden de los índices.

Vamos a ver con un ejemplo como quedaría la matriz $\mathbf{AA}$. Sea el tamaño de la matriz $\mathbf{A}$, $N \times N$ potencia de dos y la matriz $\mathbf{B}$ con dimensiones $M \times M$ también potencia de dos y con $N > M$. Para calcular $\mathbf{AuB}$ el tamaño de la matriz incógnita $\mathbf{u}$ debe de ser $N \times M$, la matriz resultante que estamos buscando sería de tamaño $NM \times NM$. Esto quiere decir que a mayor discretización de los ejes el tamaño del sistema a resolver crece exponencialmente, pero como hemos comentado ya, este método obtiene muy buenos resultados con pocos puntos, además la matriz $\mathbf{AA}$ es dispersa lo cual se sigue teniendo un coste computacional óptimo a proporción del tamaño del sistema. Para la resolución final de estos problemas se usarán métodos iterativos multigrid que están fuera del ámbito de este TFM. El sistema que se nos quedaría sería el siguiente,

$$\begin{pmatrix} & & \\ & \mathbf{AA} & \\ & & \end{pmatrix} \begin{pmatrix} u_{11} \\ \\ \vdots \\ \\ u_{NM} \end{pmatrix} = RHS \tag{4.13}$$

$\mathbf{AA}$ sería el resultado de obtener los índices de operar $\mathbf{A} \otimes \mathbf{B}$ y el vector u sería convertir la matriz incógnita $\mathbf{u}$ en un vector. Partiendo de la primera columna, colocamos las siguientes en orden de izquierda a derecha de la matriz, debajo de la anterior y así obtenemos el vector. El comando en MatLab utilizado para hacer esto es (:). También lo aplicaríamos en la parte derecha de la ecuación para que coincida el tamaño de las matrices y vectores para poder operar.

Llegados a este punto nos preguntamos como podemos obtener la expresión que nos de la solución de $\mathbf{AA}$. Hemos realizado lo siguiente. Si cuando multiplicamos dos matrices obtenemos la expresión,

$$(AB)_{(i,j)} = \sum_{l=1}^{N} A_{il} B_{lj}, \tag{4.14}$$

el problema viene cuando tenemos 3 matrices (ABC) y B es la incógnita, tendríamos que obtener un resultado tal que,

$$(ABC)_{(i,j)} = \alpha_1 u_{11} + \alpha_2 u_{12} + \alpha_3 u_{13}, + \alpha_4 u_{14} + \dots \tag{4.15}$$

y hallar que valor que tienen las α_n para encontrar la solución al problema.

A la expresión a la que se ha llegado es la siguiente,

$$(ABC)_{(i,j)} = \sum_{m=1}^{M} \sum_{l=1}^{N} A_{il} B_{lm} C_{mj}. \tag{4.16}$$

Donde el primer punto para $j = 1$ e $i = 1$ sería $\sum_{m=1}^{M} \sum_{l=1}^{N} A_{1l} B_{lm} C_{m1}$ y haciendo las cuentas se llega a obtener la matriz deseada.

El bucle realizado en MatLab para obtener la matriz **AA** a partir de las matrices obtenidas por el método de las diferencias finitas compactas, es el siguiente.

```
for i = 1:nx
    for j = 1:ny
        for l = 1:nx
            for m = 1:ny
                AA(i+(j-1)*nx,l+(m-1)*nx) = Axx(i,l) * Byy(m,j);
            end
        end
    end
end
```

Donde nx y ny son el número de discretizaciones realizadas para cada eje y A_{xx} y B_{yy} las matrices obtenidas del método CFD. Estas matrices tienen formato disperso con la siguiente distribución

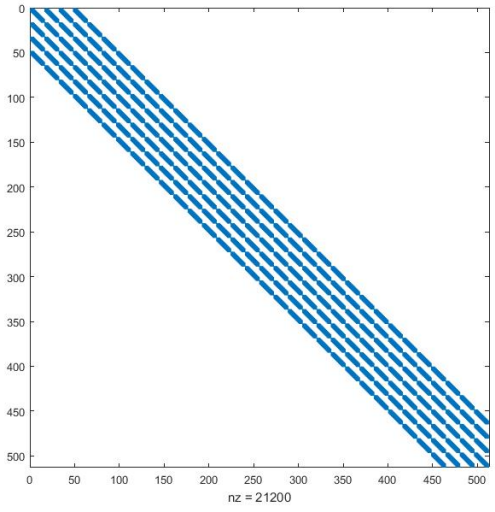

Figura 4.1: Matriz dispersa del método CFD

Esta matriz es para una discretización en el eje x de 2^4 puntos y para e eje y de 2^5 puntos, siendo del tamaño 512×512 con una cantidad de no ceros de 21,200, es

decir que hay un $91,91\,\%$ de ceros. Por tanto se pueden aplicar métodos específicos de matrices dispersas, con un enorme potencial para la reducción de memoria.

El código para obtener **AA** se ha optimizado reduciendo los puntos a calcular hasta una cuarta parte haciendo la siguiente modificación en el código.

```
for j = 1:ny
    for m = 1:ny
        AA((1:nx)+(j-1)*nx,(1:nx)+(m-1)*nx) = Axx * Byy(m,j);
    end
end
```

Trabajando con los vectores de la matriz **A** en vez de calcularlo punto a punto.

4.1.2. En la derivada primera

Para calcular la primera derivada por el método de DFC se va a calcular de manera análoga a la segunda derivada pero con las matrices proporcionadas por el método para la primera derivada, es decir, obtendremos las matrices $\mathbf{A}_x$, $\mathbf{B}_x$, $\mathbf{A}_y$, $\mathbf{B}_y$ y se nos quedará la ecuación,

$$-(\mathbf{A}_x\partial_x\mathbf{A}'_y + \mathbf{A}_x\partial_y\mathbf{A}'_y)(\mathbf{u}) = \mathbf{A}_x f(x,y)\mathbf{A}'_y. \tag{4.17}$$

Teniendo la siguiente igualdad $\mathbf{A}_x\mathbf{u} = \mathbf{B}_x\mathbf{u}_x$ para ambas direcciones tenemos,

$$\begin{array}{rcl} \mathbf{A}_x\partial_x & = & \mathbf{B}_x \\ \mathbf{A}_y\partial_y & = & \mathbf{B}_y, \end{array} \tag{4.18}$$

por lo que de la expresión 4.17 pasamos a

$$-\mathbf{B}_x\mathbf{u}(x,y)\mathbf{A}'_y - \mathbf{A}_x\mathbf{u}(x,y)\mathbf{B}'_y = \mathbf{A}_x f(x,y)\mathbf{A}'_y. \tag{4.19}$$

Ahora siguiente le mismo procedimiento que hemos aplicado en 4.10 y construyendo la matriz de tamaño $N \times M$ con la expresión 4.16, siendo N y M el número de discretizaciones para el eje x e y respectivamente, tenemos las matrices necesarias para poder calcular la solución de la ecuación convección-difusión.

Condiciones de contorno de Dirichlet

Una vez llegados a este punto, ya tenemos claro como aplicar las matrices a la ecuación para calcular las derivadas en los ejes x e y, el siguiente paso es definir correctamente las condiciones de contorno de Dirichlet, que como hemos visto en el

capítulo anterior, se trata de resolver el sistema 3.19, pero esta vez en vez de tener un vector tenemos una matriz,

$$\mathbf{u} = \begin{pmatrix} u_{11} & u_{21} & \cdots & u_{1,M-1} & u_{1M} \\ u_{21} & & & & u_{2M} \\ \vdots & & \ddots & & \vdots \\ u_{N-1,1} & & & & u_{N-1,M} \\ u_{N1} & u_{N1} & \cdots & u_{N,M-1} & u_{NM} \end{pmatrix} \tag{4.20}$$

donde vamos a tomar la primera y la última columna de $\mathbf{u}$ y la primera y última fila, excepto los valores en los bordes, que ya los tenemos en las columnas. Ahora la matriz la convertimos en vector con el comando de MatLab (:) como hemos visto en 4.13 y la posición de las $\mathbf{u}_{ij}$ que sean condiciones de contorno nos indicar las filas y columnas que tenemos que quitar de la matriz $\mathbf{AA}$ para resolver el sistema.

$$\mathbf{u} = \left(u_{11},\ldots,u_{N1}, u_{21}, \ldots, u_{N1}, \ldots, u_{1,M-1}, \ldots, u_{N,M-1}, u_{1M},\ldots,u_{NM} \right). \tag{4.21}$$

El vector que nos queda en el código realmente es la inversa de 4.21, por comodidad se ha escrito de esa forma. Lo que hacemos ahora es apuntar el número de cada fila y el número de cada columna que no deben estar en $\mathbf{AA}$ en dos vectores, que llamaremos IX e IY, lo primero que hacemos es quitar las filas de la matriz. Una vez tenemos la matriz reducida, seleccionamos el número de columnas que tenemos en IY, obteniendo una matriz que posteriormente la utilizaremos para calcular las condiciones de contorno en los bordes, multiplicando sus valores por cada u inicial correspondiente. Restaremos estos valores de las condiciones de contorno en la parte derecha de la ecuación. Con esta primera matriz reducida obtenida, ahora quitamos las columnas que se indican en el vector IY y ya se nos quedaría una matriz reducida requerida. En este caso hemos reducido la matriz de 512×512 quitando 32 filas y 14 columnas por cada lado, obteniendo una matriz de tamaño 420×420 que se va a utilizar para resolver el sistema para hallar la solución en los puntos interiores del mallado.

Esta matriz es para una discretización de $2^4 \times 2^5$ pero sin las condiciones de contorno, es decir el tamaño sería de $2^4 - 2 \times 2^5 - 2$ siendo en total 420×420 con una cantidad de no ceros de 17,028, es decir que hay un $90,35\,\%$ de ceros.

Una vez esto creamos un bucle para tener un lado derecho, con la función conocida y las condiciones de contorno correspondientes para obtener el resultado. El código que se ha utilizado para definir las condiciones de contorno conforme acabamos de explicar es el siguiente.

```
solu = sol.u(X,Y);

vectSolu = solu(:)';

% Elegimos las filas que no tiene que estar en la matriz AA y las ...
    almacenamos en IX
ix1 = (1:nx);
ix2 = linspace(nx+1,nx*ny-(2*nx-1),ny-2);
ix3 = linspace(2*nx,nx*ny-nx,ny-2);
ix4 = (nx*ny-nx+1:nx*ny);

IX = [ix1, ix2, ix3, ix4];

% Repetimos proceso para las columnas y las almacenamos en IY
IY = [iy1, iy2, iy3, iy4];

Ucc = vectSolu(IY);

AA2(IX,:) = [];      % Quitamos las filas
BB2(IX,:) = [];

RHS1 = AA2(:,IY);    % Columnas necesarias para calcular CC

AA2(:,IY) = [];      % Quitamos las columnas
BB2(:,IY) = [];

RHSCC = 0*RHS1(:,1);

for i = 1:length(IY)
    RHSCC = RHSCC + RHS1(:,i)*Ucc(i);
end
```

Ya tenemos claro como aplicar las matrices a la ecuación para calcular las derivadas en los ejes x e y. Ahora el siguiente paso es aplicar el método espectral al eje z usando Fourier.

4.2. Aplicación del método constructivo

Con lo obtenido con el método de las diferencias finitas compactas para las dos dimensiones, ahora solo tendríamos que combinarlo con el método espectral usando Fourier para la tercera dimensión y Runge-Kutta para el tiempo. Vamos a ver como lo aplicamos.

Método espectral. Fourier

Aplicamos Fourier tanto en el término lineal como en el no lineal en el eje z. Para

el término lineal se nos queda la siguiente expresión,

$$\mathcal{F}\{\mathbf{u}_{zz}\} = i^2 k_z^2 \hat{\mathbf{u}}(z) e^{ik_z 0}, \tag{4.22}$$

donde k_z es el vector que nos da los números de onda para calcular la derivada al aplicar la transformada de Fourier, el exponencial se queda elevado a cero y la parte imaginaria al cuadrado nos da el negativo quedando la transformada de Fourier de la parte lineal,

$$\mathcal{F}\{\mathbf{u}_{zz}\} = -k_z^2 \hat{\mathbf{u}}(z). \tag{4.23}$$

En la parte no lineal tenemos que aplicar la transformada a $\mathbf{u}\mathbf{u}_z$ tal que

$$\mathcal{F}\{\mathbf{u}\mathbf{u}_z\} = \hat{\mathbf{u}} i k_z \hat{\mathbf{u}}(z) e^{ik_z 0}. \tag{4.24}$$

Ahora queda aplicar el método de Runge-Kutta para realizar la discretización en el tiempo.

Método Runge-Kutta

En el capítulo anterior mostramos la expresión de los subpasos del método de Runge-Kutta en 3.36, 3.37 y 3.38, ahora aplicamos lo obtenido de las discretizaciones espaciales con sus respectivos métodos y la expresión final queda,

$$\begin{aligned}
\mathbf{u}^{n+1} \;=\; & \mathbf{u}^n + h\left[\alpha_s\left(BB\mathbf{u}^n + AA\mathbf{u}^n - A_{xx}\hat{\mathbf{u}}^n k_z^2 A'_{yy}\right) + \beta_s\left(BB\mathbf{u}^{n+1} + AA\mathbf{u}^{n+1}\right.\right. \\
& \left. - A_{xx}\hat{\mathbf{u}}^{n+1} k_z^2 A'_{yy}\right) + \gamma_s(f^n - \mathbf{u}^n B_x \mathbf{u}^n A_y - \mathbf{u}^n A_x \mathbf{u}^n B_y - \hat{\mathbf{u}}^n i k_z \hat{\mathbf{u}}^n) \\
& \left. + \zeta_{s-1}(f^{n-1} - \mathbf{u}^{n-1} B_x \mathbf{u}^{n-1} A_y - \mathbf{u}^{n-1} A_x \mathbf{u}^{n-1} B_y - \hat{\mathbf{u}}^{n-1} i k_z \hat{\mathbf{u}}^{n-1})\right].
\end{aligned} \tag{4.25}$$

Que despejando en el lado izquierdo los valores del paso siguiente y en el lado derecho los valores conocidos resulta,

$$\begin{aligned}
& \mathbf{u}^{n+1}\left[1 - h\beta_s\left(BB + AA\right)\right] + h\beta_s\left(A_{xx}\hat{\mathbf{u}}^{n+1} k_z^2 A'_{yy}\right) = \\
& \quad \mathbf{u}^n\left[1 + h\alpha_s\left(BB + AA\right) + h\gamma_s(-B_x\mathbf{u}^n A_y - A_x\mathbf{u}^n B_y)\right] - h\alpha_s A_{xx}\hat{\mathbf{u}}^n k_z^2 A'_{yy} \\
& \quad -h\gamma_s(f^n - \hat{\mathbf{u}}^n i k_z \hat{\mathbf{u}}^n) + \mathbf{u}^{n-1}\zeta_{s-1} h(-B_x\mathbf{u}^{n-1}A_y - A_x\mathbf{u}^{n-1}B_y) \\
& \quad +h\zeta_{s-1}(f^{n-1} - \hat{\mathbf{u}}^{n-1} i k_z \hat{\mathbf{u}}^{n-1}).
\end{aligned} \tag{4.26}$$

Donde el subíndice s indica el número de subpaso en el que estamos y las matrices AA y BB se han obtenido respectivamente de calcular las matrices $A_{xx}B'_{yy}$ y $B_{xx}A'_{yy}$ como se ha detallado en 4.16.

A partir de esta expresión final tenemos que crear un código introduciendo todos los parámetros de las discretizaciones para obtener la solución $\mathbf{u}$ despejando la expresión 4.26 y resolviendo el sistema.

El trabajo realizado buscando la solución del problema que acabamos de ver 4.26, ha sido realizando diferentes códigos con cada uno de los métodos vistos en el trabajo y combinándolos. Se empezó aprendiendo a programar diferencias finitas básicas aplicadas a ecuaciones en derivadas ordinarias y parciales en 1 y 2 dimensiones. Una vez dominado esto se introdujo el método de Runge-Kutta el cual se aplicó para resolver una EDP con la discretización espacial por el método de Euler implícito.

Ya con una buena base se empezó a trabajar con el método de diferencias finitas compactas en 1 dimensión hasta llegar a combinar este método con el Runge-Kutta donde ya empezábamos a obtener unos muy buenos resultados con códigos que calculaban muchos puntos en la discretización con un gasto computacional bajo y con un error que mejoraba la solución los métodos anteriores. Llegados a este punto se introdujeron los métodos espectrales, utilizando Fourier consiguiendo una gran capacidad computacional paras calcular las derivadas tanto en términos lineales como en no lineales.

Familiarizado con los tres métodos, se empezó a programar problemas en dos direcciones combinando dichos métodos llegando a conseguir dos grandes resultados entre todos los obtenidos. uno ha sido hacer un código combinado los métodos de CDF y Fourier para las direcciones x e y con términos no lineales, con la discretización en el tiempo por el método de euler implícito. Y el mayor logro del trabajo ha sido resolver el laplaciano, $\Delta\mathbf{u}$ con el método de diferencias finitas compactas para 2D. Los resultados los tenemos en el siguiente capítulo.

Capítulo 5

Resultados

Los resultados que vamos a ver a continuación son los obtenidos en los cuatro grandes avances que se han ido haciendo durante el proceso de aprendizaje de programación.

El primero muestra la solución de la ecuación $\mathbf{u}_t + \mathbf{u}\mathbf{u}_x - \mathbf{u}_{xx} = f(x,t)$ por el método de Runge-Kutta con tres subpasos. A continuación vamos a ver como el resultado del código, en azul, aproxima muy bien a la solución de la ecuación, en rojo, y el error que nos da muy bueno, 10^{-6}, respecto al número de puntos que cogemos en la discretización.

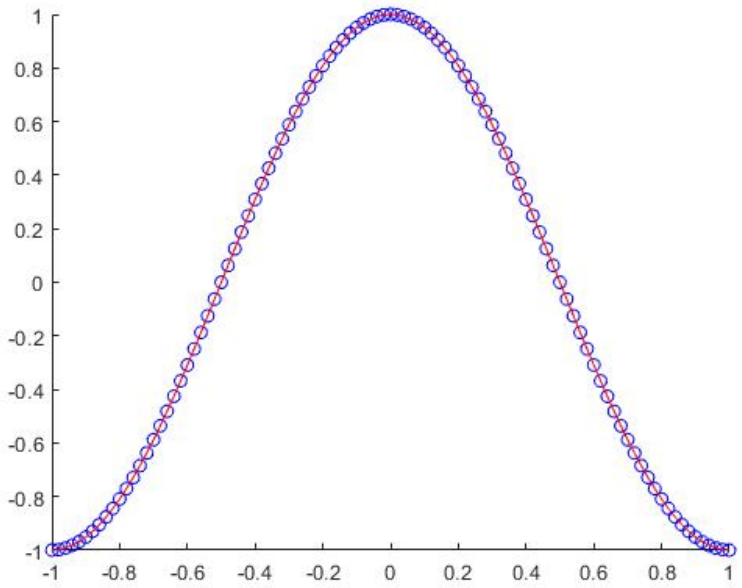

Figura 5.1: Método Runge-Kutta

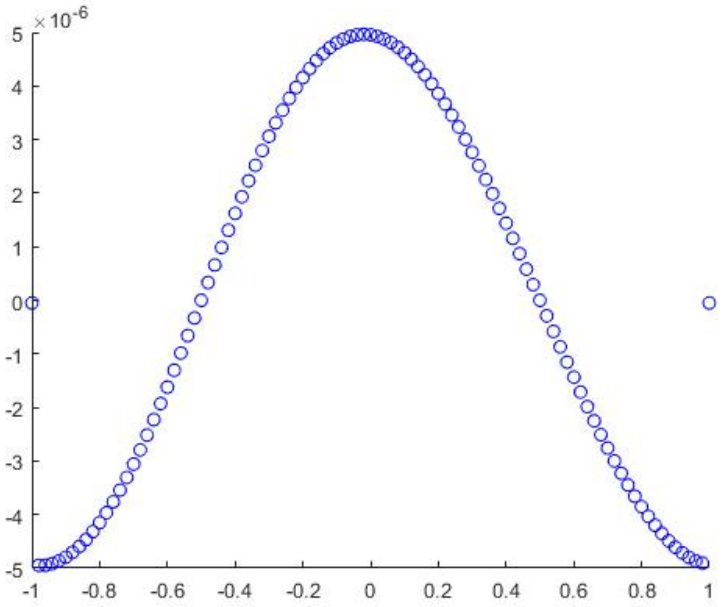

Figura 5.2: Método Runge-Kutta - Error

El siguiente avance importante que se hizo, fue hacer el código con el método de diferencias finitas compactas para realizar la discretización en el espacio combinado con Runge-Kuta discretizando el tiempo, para obtener el resultado de la ecuación $\mathbf{u}_t + \mathbf{u}\mathbf{u}_x - \mathbf{u}_{xx} = f(x,t)$.

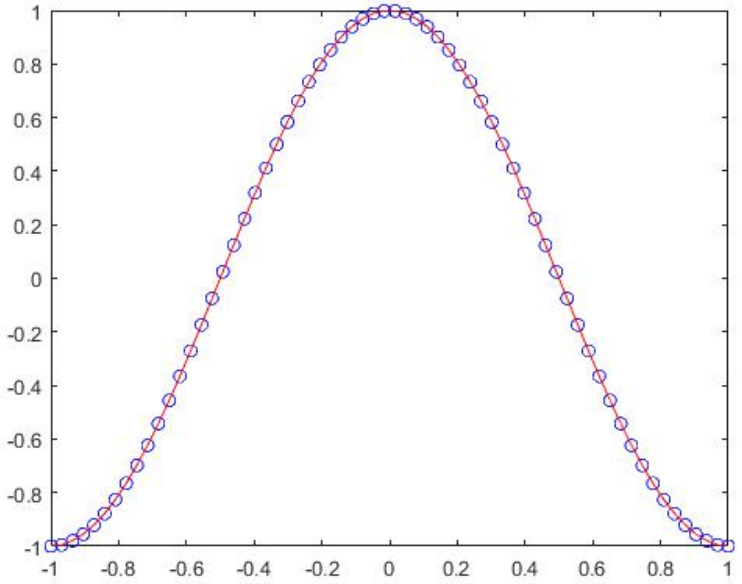

Figura 5.3: Método diferencias finitas compactas en 1D

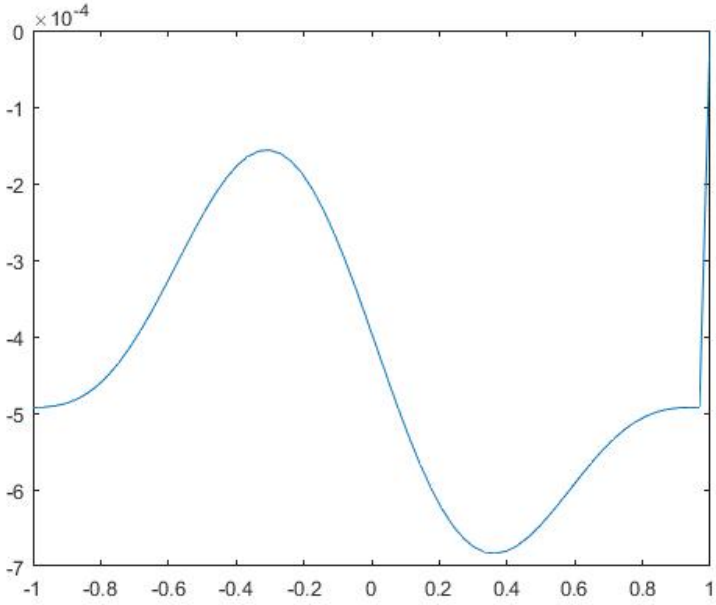

Figura 5.4: Método diferencias finitas compactas en 1D - Error

Siguiendo con el aprendizaje de los métodos, combinamos el método espectral con el de las diferencias finitas compactas aplicados al eje x e y respectivamente. En este código empezamos a utilizar las condiciones de contorno, porque hasta ahora se había trabajado con funciones que la solución en la frontera era igual a 0.

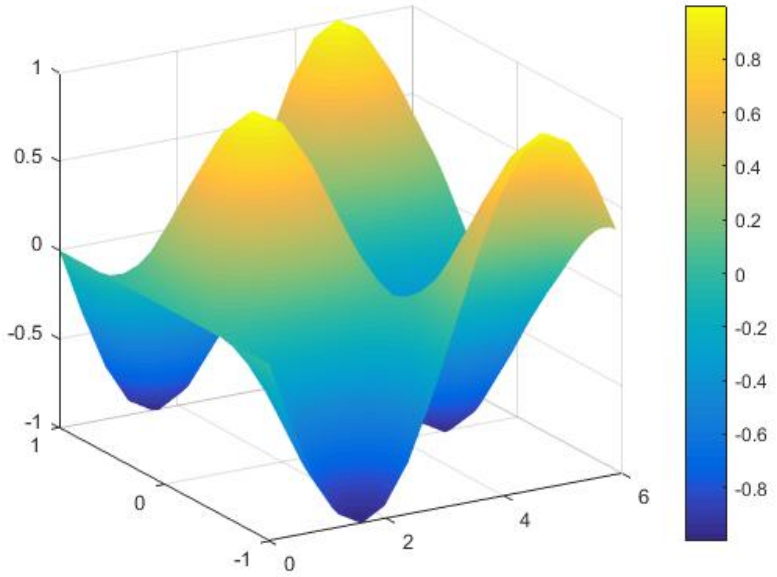

Figura 5.5: Solución con método espectral y CFD

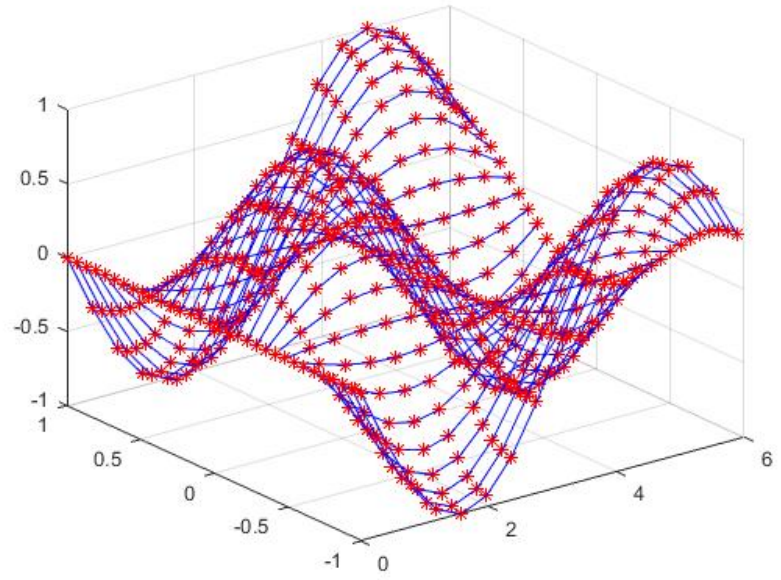

Figura 5.6: Método espectral y CFD - Aprovimación de la solución

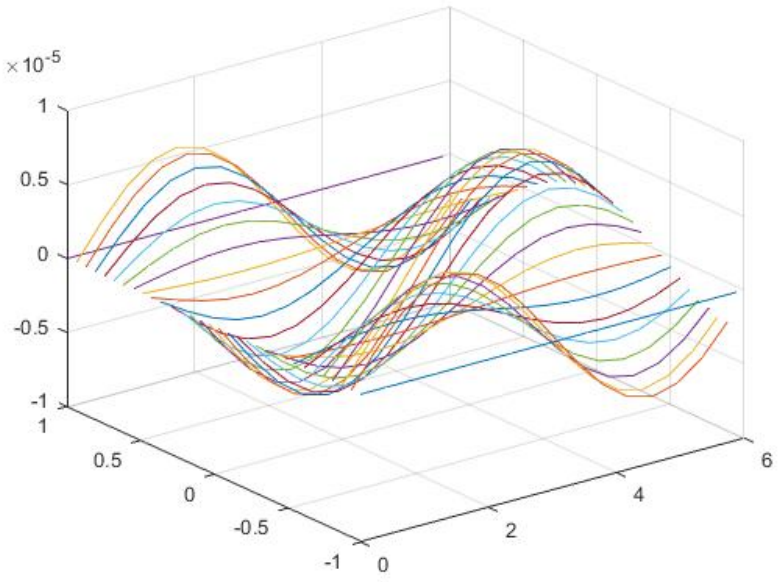

Figura 5.7: Método espectral y CFD - Error

Por último y el resultado más importante que hemos obtenido en el trabajo ha sido encontrar la solución del laplaciano en 2D por el método de diferencias finitas compactas, donde la dificultad residía en definir adecuadamente la matriz interior para

obtener un buen resultado, además de ubicar las condiciones de contorno de Dirichlet correctamente.

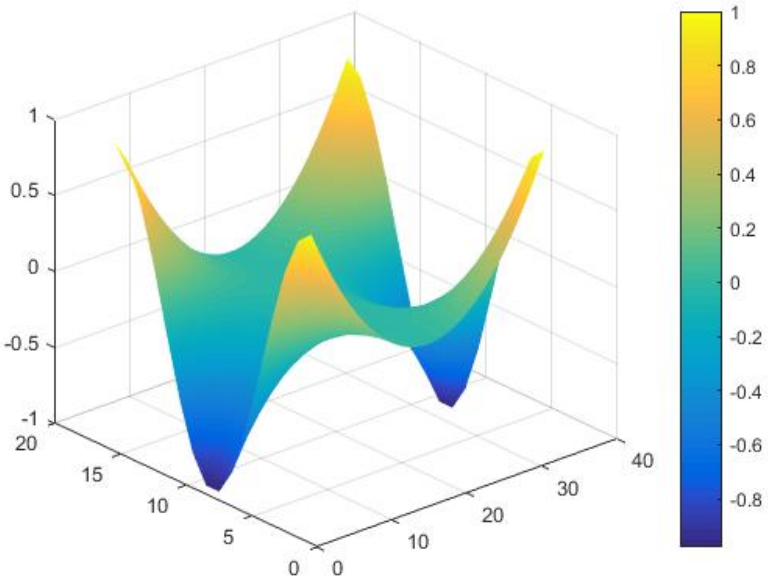

Figura 5.8: Método espectral y CFD 2D

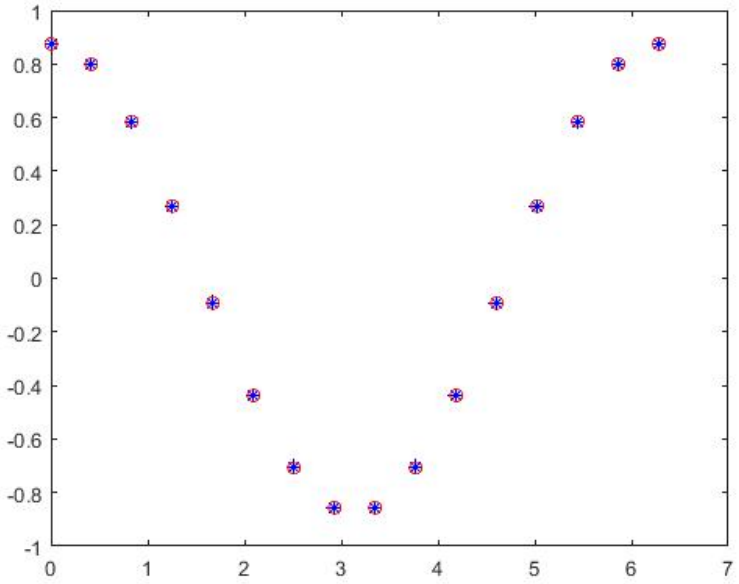

Figura 5.9: Método espectral y CFD 2D - Aproximación a la solución

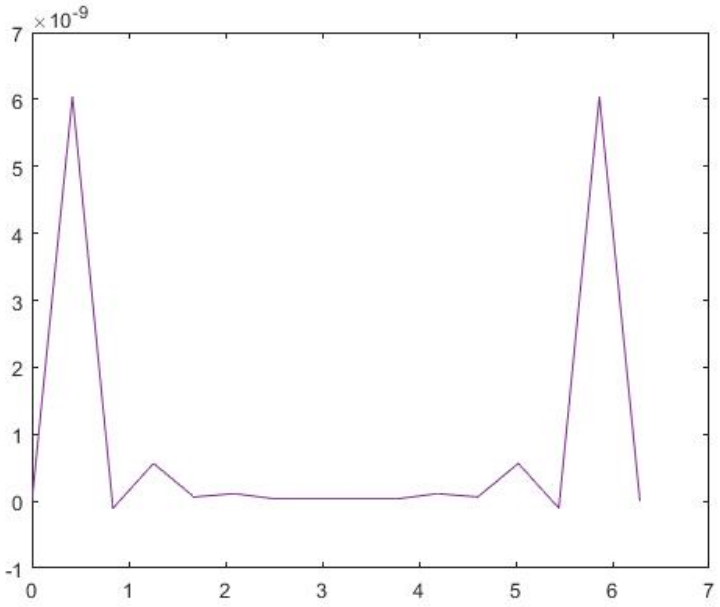

Figura 5.10: Método espectral y CFD 2D - Error

Errores obtenidos en los métodos utilizados

MÉTODO	Error
Runge-Kutta	10^{-6}
Runge-Kutta y CFD	10^{-4}
CFD y Fourier	10^{-5}
CFD 2D	10^{-9}

Capítulo 6

Conclusión

Los métodos numéricos computacionales son una herramienta muy potente para encontrar la solución numérica a problemas que son muy costosos de calcular. En base a los objetivos planteados del trabajo, se pueden extraer las siguientes conclusiones. Los métodos vistos en este proyecto han dado muy buenos resultado con un coste computacional bajo y a medida que los combinábamos los resultados mejoraban notablemente. Desde el punto de vista de la investigación se han obtenido buenos resultados como resolver el Laplaciano en dos dimensiones con el método de diferencias finitas compactas. Y desde el punto de vista del estudiante se han adquirido grandes habilidades para manejar las herramientas computacionales y aplicar los métodos numéricos utilizados para resolver problemas en derivadas parciales. Además de adquirir una gran capacidad de análisis sobre lo que se está trabajando para encontrar errores y hacer mejoras consiguiendo así un código más óptimo y depurado.

Como futuro trabajo habría que seguir construyendo el esquema numérico hasta llegar a encontrar la solución deseada de la ecuación 4.26 obteniendo el código test que se plantea en la introducción del trabajo. Lo siguiente que habría que hacer a partir de donde nos hemos quedado sería aplicar el método de diferencias finitas compactas en 2D a una ecuación con término no lineal y ver como responde en el tiempo. Una vez tengamos esto pasaríamos a aplicar el método de Runge-Kutta para conseguir velocidad de computación y precisión en el resultado. Después habría que empezar con los problemas en tres dimensiones resolviendo dos direcciones con el método CFD y la tercera dimensión con Fourier, como detalla el esquema visto en el capítulo 2, cuando se consiga resolver el problema de convección-difusión con este esquema aplicaríamos Runge-Kutta y ya resolveríamos el problema planteado 4.26 en el capítulo 4.

Una vez se haya conseguido el código planteado, el proyecto se culmina adaptando el programa para resolver sistemas de ecuaciones como el visto en 2.1. Consiguiendo así un código que se pueda aplicar a las ecuaciones de Navier-Stokes para recoger datos aerodinámicos fiables que es el objetivo principal de este proyecto realizado en el IUMPA.

ANEXO I. Códigos de MatLab

Código con el método Runge-Kutta con diferencias finitas implícito en el eje x, para resolver EDP no lineal.

```
% Solucion EDP ut-uxx+uux = f con metodo RK implicito

% Mallado

x0 = -1d0;
xf =  1d0;
N  =  101;

x  = linspace(x0,xf,N)';
Dx = (xf-x0)/(N-1);


% Discretizacion en tiempo

t0 = 0d0;
tf = 5d-6;
M = 101;

t = linspace(t0,tf,M);
Dt = (tf-t0)/(M-1);


% Solucion y lado derecho

u   = @(x,t)  cos(pi*x)*exp(-t);
ut  = @(x,t) -cos(pi*x)*exp(-t);
uxx = @(x,t) -pi^2*cos(pi*x)*exp(-t);
ux  = @(x,t) -pi*sin(pi*x)*exp(-t);

fun = @(x,t) ut(x,t) - uxx(x,t) + u(x,t).*ux(x,t);


% Llamamos a la funcion

[¬,sol] = spalartEDP1(fun,x,Dx,t,Dt,u);


```

```
38 % Plot
39
40 ind = 1;
41 figure(ind); clf;  hold on; ind = ind+1;
42 % Resultados
43 plot(x,sol(:,end),'bo',x,u(x,t(end)),'r');
44
45 figure(ind); clf;  hold on; ind = ind+1;
46 % Error
47 plot(x,sol(:,end) - u(x,t(end)), 'bo');
48
49
50 function [t,u] = spalartEDP1(fun,x,Dx,t,Dt,ufun)
51
52 ww = Dt/Dx^2;
53 w  = Dt/Dx;
54 h  = Dt;
55 n  = length(x);
56 m  = length(t);
57
58
59 % Parametros del metodo
60 alfa  =  [29/96    -3/40   1/6];
61 beta  =  [37/160   5/24    1/6];
62 gamma =  [8/15     5/12    3/4];
63 xi    =  [0       -17/60   -5/12];
64
65
66 % Creamos matrices para obtener las derivadas
67 v1 = -2*ones(1,n);
68 v2 = ones(1,n-1);
69 v3 = ones(1,n);
70
71 A  = diag(v1) + diag(v2,1) + diag(v2,-1);
72 B  = diag(v3) - diag(v2,-1);
73
74
75 % Metodo Runge-Kutta
76 u = zeros(n,m);
77 u(:,1) = ufun(x,0);
78
79 p = zeros(1,3); q = zeros(1,3); r = zeros(1,3);
80 rr = zeros(1,3); s = zeros(1,3); ss = zeros(1,3);
81
82 I = eye(n,m);
83 M = zeros(n,m,3);
84
85 for j=1:3
86     p(j)  = beta(j)*ww;
87     q(j)  = alfa(j)*ww;
88     r(j)  = gamma(j)*h;
89     rr(j) = gamma(j)*w;
```

```
90      s(j)  = xi(j)*h;
91      ss(j) = xi(j)*w;
92      M(:,:,j) = inv(I - p(j)*A);
93  end
94
95
96  for i = 1:m-1
97      uold  = u(:,1);
98      uold2 = uold;  % Notice that ss(1) = 0
99
100     for j = 1:3
101
102         fun = feval(fun,x,t(i)+(j-1)/3*h);
103
104         duxx = A*uold;
105         dux  = B*uold;
106         dux2 = B*uold2;
107
108         NL = rr(j)*uold.*dux + r(j)*fun + ss(j)*uold2.*dux2 + ...
                s(j)*fun;
109
110         RHS = uold + q(j)*duxx + NL;     % Right hand side
111
112         unew =  M(:,:,j) * RHS;          % Resolvermos sistema
113
114         % Actualizo
115         uold2 = uold;
116         uold  = unew;
117
118     end
119
120     unew(1)   = ufun(x(1),t(i));
121     unew(end) = ufun(x(end),t(i));
122
123     u(:,i+1) = unew;
124
125 end
```

Código con el método DFC en 1D para el eje x y Runge-Kutta para la discretización de t, para una EDP no lineal.

```
1  % EDP no lineal ut + uux - uxx = f(x,t) con CFD y RK
2
3  % El addpath para llamar a las matrices
4  if isunix
5  root = '/home/sergio/Dropbox/';
6  else
7  root='C:\Users\Lenovo\Dropbox\';
8  end
9
10 addpath(strcat(root,'tfmPeinado/derivatives'));
```

```
11
12 % Discretizacion de x
13
14 x0 = -1d0;
15 xf = 1d0;
16 nx = 2^6;
17 x  = linspace(x0,xf,nx);
18 dx = (xf-x0) / (nx-1);
19
20 derivada = 1;
21 [fStencil,nfStencil] = getStencil(5,5);
22 [Ax,Bx,¬,¬] = matrixCFD(x,fStencil,nfStencil,derivada,1);
23
24 derivada = 2;
25 [fStencil,nfStencil] = getStencil(7,7);
26 [Axx,Bxx,¬,¬] = matrixCFD(x,fStencil,nfStencil,derivada,2);
27
28 % Discretizacion de t
29
30 t0 = 0d0;
31 tf = 1d-4;
32 nt = 2^9;
33 t  = linspace(t0,tf,nt);
34 dt = t(2) - t(1);
35
36 % Solucion
37
38 sol.u   = @(x,t)  cos(pi*x).*exp(-t);
39 sol.ut  = @(x,t) -cos(pi*x).*exp(-t);
40 sol.ux  = @(x,t) -pi*sin(pi*x).*exp(-t);
41 sol.uxx = @(x,t) -pi^2*cos(pi*x).*exp(-t);
42
43 sol.f = @(x,t) sol.ut(x,t) - sol.uxx(x,t) + sol.u(x,t).*sol.ux(x,t);
44
45
46 %% Solucion numerica
47
48 I = eye(nx,nx);
49 M = zeros(nx,nx,3);
50
51
52 % Parametros del metodo RK
53 p  = zeros(1,3); q = zeros(1,3); r = zeros(1,3);
54 rr = zeros(1,3); s = zeros(1,3); ss = zeros(1,3);
55
56
57 alfa  =  [29/96    -3/40   1/6];
58 beta  =  [37/160   5/24    1/6];
59 gamma =  [8/15     5/12    3/4];
60 xi    =  [0        -17/60   -5/12];
61
62 for j=1:3
```

```
63 p(j)  = beta(j)*dt;
64 q(j)  = alfa(j)*dt;
65 r(j)  = gamma(j)*dt;
66 s(j)  = xi(j)*dt;
67
68 M(:,:,j) = inv(Axx - p(j)*Bxx);
69 end
70
71
72 unew = sol.u(x,t0);
73
74 uold  = unew;
75 uold2 = uold;
76
77
78 for i = 1:nt-1
79
80 dxuold  = Ax\(Bx*uold');
81 dxuold2 = Ax\(Bx*uold2');
82
83 for j = 1:3
84
85 fun = feval(sol.f,x,t(i)+(j-1)/3*dt);
86
87 NL = r(j) * (fun - uold.*dxuold) + ...      % Parte no lineal
88 s(j) * (fun - uold2.*dxuold2);
89
90 RHS = (Axx + q(j) * Bxx) * uold' + Axx*NL;    % Right hand side
91
92 unew =  M(:,:,j) * RHS;                      % SISTEMA DE ECUACIONES
93
94  % Actualizo
95 uold2 = uold;
96 uold  = unew;
97
98 end
99
100 unew(1)   = sol.u(x(1),t(i+1));         % Condiciones contorno
101 unew(end) = sol.u(x(end),t(i+1));
102
103 end
104
105
106  %% Plot
107
108 figure(1)
109 plot(x,unew(:,end),'bo', x,sol.u(x,tf)','r')
110
111 figure(2)
112 plot(x, unew(:,end) - sol.u(x,tf)')
```

Código con el método DFC para el eje y, Fourier en el eje x.

```
% Resolvemos la EDP ut - (uxx + uyy) = f(x,y,t) con Fourier para ...
   el eje 'x'
% y diferencias finitas compactas CFD

% El addpath para llamar a las matrices
if isunix
    root = '/home/sergio/Dropbox/';
    else
    root='C:\Users\Lenovo\Dropbox\';
end

addpath(strcat(root,'tfmPeinado/derivatives'));


% Discretizacion de x
%
N  = 2^4;            % Potencia de 2 o 3
dx = 2*pi/N;         % Trabajamos en intervalor 2*pi
x  = 0:dx:2*pi-dx;   % No tomamos 2*pi por no tener sobre-determinacion

k = [0:N/2-1, 0, -N/2+1:-1];     % Vector de elementos k que de la DFT
nk = length(k);

% Discretizacion de y

y0 = -1;    y1 = 1;
M = 2^5;
y = linspace(y0,y1,M);
dy = y(2)-y(1);

derivada = 2;
[fStencil,nfStencil] = getStencil(7,7);
[Ayy,Byy,¬,¬] = matrixCFD(y,fStencil,nfStencil,derivada,2);

% Discretizacion en el tiempo

t0 = 0d0;
tf = 1d-4;
dt = 1e-5;


% solucion de la EDP

sol.u   = @(x,y,t)  sin(x).*cos(pi*y).*exp(-t);
sol.ut  = @(x,y,t) -sin(x).*cos(pi*y).*exp(-t);
sol.uxx = @(x,y,t) -sin(x).*cos(pi*y).*exp(-t);
sol.uyy = @(x,y,t) -pi^2*sin(x).*cos(pi*y).*exp(-t);

sol.F = @(x,y,t) sol.ut(x,y,t) - (sol.uxx(x,y,t) + sol.uyy(x,y,t));
```

```
49
50
51  % Malla
52  [X,Y] = meshgrid(x,y);
53  X = X'; Y = Y';
54
55  unew = zeros(nk,M)';
56
57  I = eye(M);
58
59
60  while t0 ≤ tf
61
62      ufft = fft(sol.u(X,Y,t0))';
63
64      fF   = fft(feval(sol.F,X,Y,t0))';
65
66      % Condiciones de contorno
67      unew(1,   :) = ufft(1,   :);
68      unew(end,:) = ufft(end,:);
69
70      RHS = Ayy*(ufft + dt*fF);
71
72      for wn = 1:nk
73
74          LHS = Ayy + dt*Ayy*k(wn)^2 - dt*Byy;
75
76          B = LHS(2:M-1,2:M-1);
77
78          b = RHS(2:M-1,wn);
79
80          CC1   = ufft(1   ,wn) * LHS(2:M-1,   1);    % Condiciones de ...
                contorno
81          CCend = ufft(end,wn) * LHS(2:M-1,end);
82
83          unew(2:M-1,wn) =  B \ (b - CC1 - CCend) ;
84
85      end
86
87      t0 = t0+dt;
88
89  end
90
91  % Volvemos a fisico
92
93  uPhy = ifft(unew');
94
95  % Plot
96
97  figure(1)
98  plot3(X,Y, uPhy,'r', X,Y, sol.u(X,Y,tf),'b')
99
```

```
figure(2)
plot3(X,Y, uPhy - sol.u(X,Y,tf))

figure(3)
plot3(X,Y, uPhy,'r')
```

Código con el método de diferencias finitas compactas en 2D para resolver $\Delta \mathbf{u}$.

```
%% Resolucion ecucacion lineal con metodo DFC 2D : -uxx - uyy = f(x,t)

% Addpath para llamar a las matrices del metodo CFD
if isunix
root = '/home/sergio/Dropbox/';
else
root='C:\Users\Lenovo\Dropbox\';
end

addpath(strcat(root,'tfmPeinado/derivatives'));

% Discretizacion en x

x0 = 0d0;
xf = 2*pi;
nx = 2^4;
x  = linspace(x0,xf,nx);
dx = x(2) - x(1);

derivada = 2;
[fStencil,nfStencil] = getStencil(7,7);
[Axx,Bxx,¬,¬] = matrixCFD(x,fStencil,nfStencil,derivada,2);

% Discretizacion en y

y0 = -1d0;
yf = 1d0;
ny = 2^5;
y = linspace(y0,yf,ny);
dy = y(2) - y(1);

derivada = 2;
[fStencil,nfStencil] = getStencil(7,7);
[Ayy,Byy,¬,¬] = matrixCFD(y,fStencil,nfStencil,derivada,2);


% Calculamos las inversas para evitarlas dentro el bucle

Byy = Byy';
Ayy = Ayy';

% Mallado

```

```
44 [X,Y] = meshgrid(x,y);
45 X = X'; Y = Y';
46
47
48 %% Solucion analitica
49
50 sol.u   = @(x,y)  cos(x).*(y.^2);
51 sol.uxx = @(x,y) -cos(x).*(y.^2);
52 sol.uyy = @(x,y)  2*cos(x);
53
54 sol.f = @(x,y) - sol.uxx(x,y) - sol.uyy(x,y);
55
56
57 %% MATRICES DEL METODO CFD
58
59 AA = zeros(nx*ny);
60 BB = zeros(nx*ny);
61
62 for j = 1:ny
63     for m = 1:ny
64         AA((1:nx)+(j-1)*nx,(1:nx)+(m-1)*nx) = Axx * Byy(m,j);
65         BB((1:nx)+(j-1)*nx,(1:nx)+(m-1)*nx) = Bxx * Ayy(m,j);
66     end
67 end
68
69 %% SOLUCION NUMERICA
70
71 solu = sol.u(X,Y);
72
73 vectSolu = solu(:)';
74
75 % Filas y columnas que quitas de AA y BB para calcular solucion ...
      interior
76 ix1 = (1:nx);
77 ix2 = linspace(nx+1,nx*ny-(2*nx-1),ny-2);
78 ix3 = linspace(2*nx,nx*ny-nx,ny-2);
79 ix4 = (nx*ny-nx+1:nx*ny);
80 IX = [ix1, ix2, ix3, ix4];
81
82 iy1 = (1:nx);
83 iy2 = linspace(nx+1,nx*ny-(2*nx-1),ny-2);
84 iy3 = linspace(2*nx,nx*ny-nx,ny-2);
85 iy4 = (nx*ny-nx+1:nx*ny);
86 IY = [iy1, iy2, iy3, iy4];
87
88 Ucc = vectSolu(IY);
89
90 AA(IX,:) = [];      % quitamos las filas
91 BB(IX,:) = [];
92
93 RHS1 = AA(:,IY) + BB(:,IY);    %  Columnas necesarias para calcular ...
      CC de Dirichlet
```

```
94
95  AA(:,IY) = [];        % quitamos las columnas
96  BB(:,IY) = [];
97
98  RHSCC = 0*RHS1(:,1);
99
100 % Condiciones de contorno de Dirichlet que restamos en RHS
101 for i = 1:length(IY)
102     RHSCC = RHSCC + RHS1(:,i)*Ucc(i);
103 end
104
105
106 % Sistema a resolver
107 RHS  = Axx * sol.f(X,Y) * Ayy;
108
109 RHSI = RHS(2:end-1,2:end-1);      % right hand side INTERIOR
110
111 LHS  = - BB - AA;                 % left hand side
112
113 % Resultado
114 u_int = LHS \ sparse(RHSI(:) + RHSCC);
115
116 % Solucion final
117 u = zeros(nx,ny);
118
119 u(:,  1) = solu(:,  1);      % Condiciones de contorno
120 u(:,end) = solu(:,end);
121
122 u(1,  2:end-1) = solu(1,  2:end-1);
123 u(end,2:end-1) = solu(end,2:end-1);
124
125 nx2 = nx -2;
126 ny2 = ny -2;
127
128 u(2:end-1,2:end-1) = reshape(u_int,nx2,ny2);
129
130
131 %% Plot
132 figure(1)
133 plot(X,u(:,end-1),'b*',X,solu(:,end-1),'ro')
134
135 figure(2)
136 plot(X,u,'b',X,solu,'r')
137
138 figure(3)
139 plot(X, u(:,end-1) - solu(:,end-1))
```

ANEXO II. Presupuesto

Se va a valorar monetariamente el trabajo invertido en el desarrollo del presente proyecto. Tendremos en consideración las horas dedicadas por el personal implicado en el proyecto así como los recursos materiales. Como unidad para valorar el esfuerzo invertido por el personal será la hora de trabajo y la unidad monetaria los euros por hora trabajada. Para los materiales empleados se empleará los euros.

Nº orden	Unidad	Cantiad	Descripción	Precio	Importe
01	horas	540	Coste del ingeniero	12 €	6480 €
02	horas	40	Coste del tutor	35 €	1.400 €
03	año	1	Licencia MatLab	5.000 €	5.000 €
04	Ud.	1	PC del alumno	1.000 €	1.000 €
05	Ud.	1	Material de oficina	40 €	40 €
06			Total presupuesto		13.920 €

El coste bruto total del proyecto es 13.920 €. Aplicando los pertinentes presupuestos y gastos indirectos

Nº orden	Resumen precios	Importe
1	Coste bruto proyecto	13.920 €
	20 % de Gastos indirectos	2.784 €
	6 % de Beneficio industrial	836 €
Total presupuesto antes de impuesto		17.553 €
	21 % I.V.A.	3.686 €

El coste neto total estimado del proyecto es

Total presupuesto IVA incluido	**21.239 €**

El importe total al que asciende el presupuesto es de **VEINTIÚN MIL DOSCIENTOS TREINTA Y NUEVE**.

Bibliografía

[1] George Allen Baker and John L Gammel. *The Padé approximant in theoretical physics*. Academic Press, 1970.

[2] E Oran Brigham and RE Morrow. The fast fourier transform. *IEEE spectrum*, 4(12):63–70, 1967.

[3] Claudio Canuto, M Yousuff Hussaini, Alfio Quarteroni, A Thomas Jr, et al. *Spectral methods in fluid dynamics*. Springer Science & Business Media, 2012.

[4] Lothar Collatz. *The numerical treatment of differential equations*, volume 60. Springer Science & Business Media, 2012.

[5] A. George and J.W.H. Liu. *Computer solution of large sparse positive definite systems*. Prentice-Hall series in computational mathematics. Prentice-Hall, 1981.

[6] David Gottlieb, Steven A Orszag, and GA Sod. Numerical analysis of spectral methods: Theory and application (cbms-nsf regional conference series in applied mathematics), 1978.

[7] David Gottlieb, Steven A Orszag, and GA Sod. Numerical analysis of spectral methods: Theory and application (cbms-nsf regional conference series in applied mathematics), 1978.

[8] J.D. Lambert. *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*. Wiley, 1991.

[9] Sanjiva K. Lele. Lele, s.k.: Compact finite difference schemes with spectral-like resolution. journal of computational physics 103(1), 16-42. *Journal of Computational Physics*, 103:16–42, 11 1992.

[10] Philippe R Spalart. Theoretical and numerical study of a three-dimensional turbulent boundary layer. *Journal of Fluid Mechanics*, 205:319–340, 1989.

[11] Philippe R Spalart, Robert D Moser, and Michael M Rogers. Spectral methods for the navier-stokes equations with one infinite and two periodic directions. *Journal of Computational Physics*, 96(2):297 – 324, 1991.

[12] Lloyd N Trefethen. Fourier analysis of numerical approximations of hyperbolic equations (r. vichnevetsky and jb bowles). *SIAM Review*, 26(3):439–441, 1984.